NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EVOLUTION OF A GRAPHICAL USER INTERFACE FOR THE RAPID PROTOTYPING OF REAL-TIME EMBEDDED SYSTEMS

by

Kenneth Brett Moeller

September 1997

Advisors: Man-Tak Shing
Valdis Berzins

Approved for public release; Distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1997	Master's Thesis

4. TITLE AND SUBTITLE
EVOLUTION OF A GRAPHICAL USER INTERFACE FOR THE RAPID PROTOTYPING OF REAL-TIME EMBEDDED SYSTEMS

5. FUNDING NUMBERS

6. AUTHOR(S)
Moeller, Kenneth Brett

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Computer-Aided Prototyping System (CAPS) is an integrated collection of software tools that support the development of software systems utilizing the prototype paradigm. Central to CAPS is the Prototype System Description Language (PSDL). The PSDL Editor supplied in CAPS Release 1 provided a unique combination of a graphical interface for editing PSDL data flow diagrams and an attribute-grammar based text editor to enforce syntactically correct PSDL prototypes. Feedback from CAPS users highlighted on productivity impacts due to the dual user interface as well as the steep learning curve required to become proficient with the attribute-grammar based text editor.

This research initiates the development of the next generation of the CAPS PSDL Editor, focusing on the graph editor. Our approach provides a single graphical user interface with pull-down menus for editing both graphical and text information. Automatic syntax generation and validation as well as limited semantic validation is provided by a background syntax/semantics checker. The result of this research is a working graph editor meeting all the new requirements. When integrated with a the new syntax/semantics checker, CAPS Release 2 will have a PSDL Editor with enhanced capabilities and expected productivity improvements.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Syntax Directed Editor, User Interface, Rapid Prototyping	452
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

EVOLUTION OF A GRAPHICAL USER INTERFACE FOR THE RAPID PROTOTYPING OF REAL-TIME EMBEDDED SYSTEMS

Kenneth Brett Moeller
B.S., University of California at San Diego, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

ABSTRACT

The Computer-Aided Prototyping System (CAPS) is an integrated collection of software tools that support the development of software systems utilizing the prototype paradigm. Central to CAPS is the Prototype System Description Language (PSDL). The PSDL Editor supplied in CAPS Release 1 provided a unique combination of a graphical interface for editing PSDL data flow diagrams and an attribute-grammar based text editor to enforce syntactically correct PSDL prototypes. Feedback from CAPS users highlighted on productivity impacts due to the dual user interface as well as the steep learning curve required to become proficient with the attribute-grammar based text editor.

This research initiates the development of the next generation of the CAPS PSDL Editor, focusing on the graph editor. Our approach provides a single graphical user interface with pull-down menus for editing both graphical and text information. Automatic syntax generation and validation as well as limited semantic validation is provided by a background syntax/semantics checker. The result of this research is a working graph editor meeting all the new requirements. When integrated with a the new syntax/semantics checker, CAPS Release 2 will have a PSDL Editor with enhanced capabilities and expected productivity improvements.

DISCLAIMER

The computer programs provided in the appendices are supplied on an "as is" basis, with no warrantees of any kind. The author bears no responsibility for any consequences of using these program.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Just as the development of the PSDL Editor has depended on many individuals, so has my education at the Naval Postgraduate School. Special thanks to my thesis advisors, Professors Man-Tak Shing and Valdis Berzins. But I would never have made it here without my mother and father. Nor would any of my stuff made it to Monterey if not for my sister, Karen, who packed my house. Lisa Fung and the Brown Haired Girl provided me with encouragement. And God opened every door. Thanks to all.

I would also like to thank the Linux Users Group at the Naval Postgraduate School. This entire thesis was written on a Linux platform using LaTeX. Figures used in this document were produced with XFig, XV, MetaPost, and daVinci. I would recommend that all Computer Science students become active in the Linux Users Group.

I. INTRODUCTION

A. BACKGROUND

The Computer-Aided Prototyping System (CAPS) [Ref. 1] is an integrated collection of software tools that support the development of software systems using the prototyping paradigm. The focus of CAPS is the development of hard real-time embedded software systems [Ref. 2]. Through the use of the prototyping paradigm, CAPS is especially well suited to support the development and validation of system requirements as well as feasibility studies [Ref. 3].

The software tools that comprise CAPS are organized into four major subsystems: Editors, Software Base, Execution Support, and Project Control (see Figure 1) [Ref. 4]. Underlying each of these subsystems is the Prototype System Description Language (PSDL) [Ref. 3].

Prototypes developed in CAPS are specified using PSDL. PSDL is a high-level language, rich in abstraction and with facilities for capturing the requirements

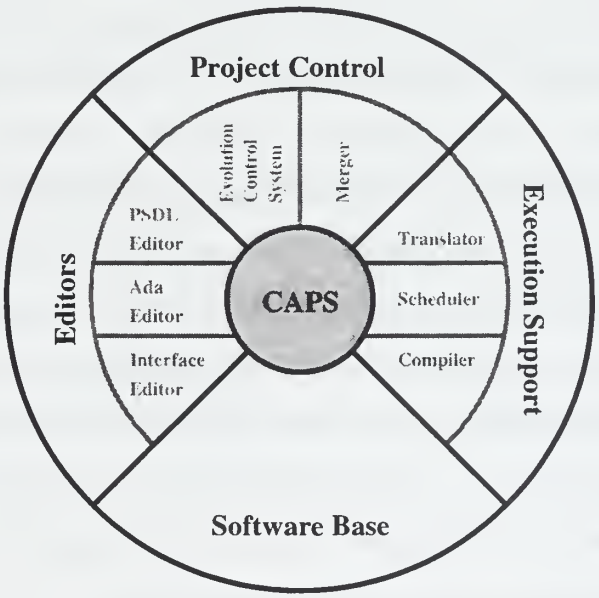


Figure 1. CAPS Subsystems, After [Ref. 4]

of real-time systems. In PSDL, a prototype is partially specified as an augmented data flow diagram. A representation of a PSDL prototype is provided in Figure 2. The augmented data flow diagram is depicted by the shaded box at the top of the figure.

A data flow diagram consists of a network of operators which communicate through data streams. The data flow diagram depicted in Figure 2 is composed of three operators (represented as circles) and two data streams (represented as the directed-lines connecting the circles within the shaded box). The data flow diagram is augmented with timing and control constraints. In Figure 2, each operator is assigned a Maximum Execution Time (MET), which can be found at the bottom-right of each operator. In this example, all MET values are in milliseconds (ms).

A data flow diagram is one element of a PSDL component. PSDL provides two kinds of components: abstract state machines and abstract data types. An augmented data flow diagram is a visual abstraction of the processing to be performed by an operator. Operators are introduced through PSDL components. An abstract state machine specifies a single operator. Multiple operators can be introduced as part of an abstract data type. The prototype itself is implemented as an operator defined by an abstract state machine. There are two parts to each component: a specification and an implementation. The specification defines the component's interface. The component's implementation can be realized by either decomposing the component into a more detailed data flow diagram or by a reference to a programming language implementation. Figure 2 depicts the top level data flow diagram of a robot prototype. In this example, each operator has been implemented with a reference to a programming language implementation, illustrated by the three boxes below the augmented data flow diagram. The current release of CAPS supports two programming language implementations. An operator can be implemented as an Ada package or as a TAE module, providing a graphical interface. [Ref. 5]

The Editor subsystem provides a collection of editors which are tailored to the

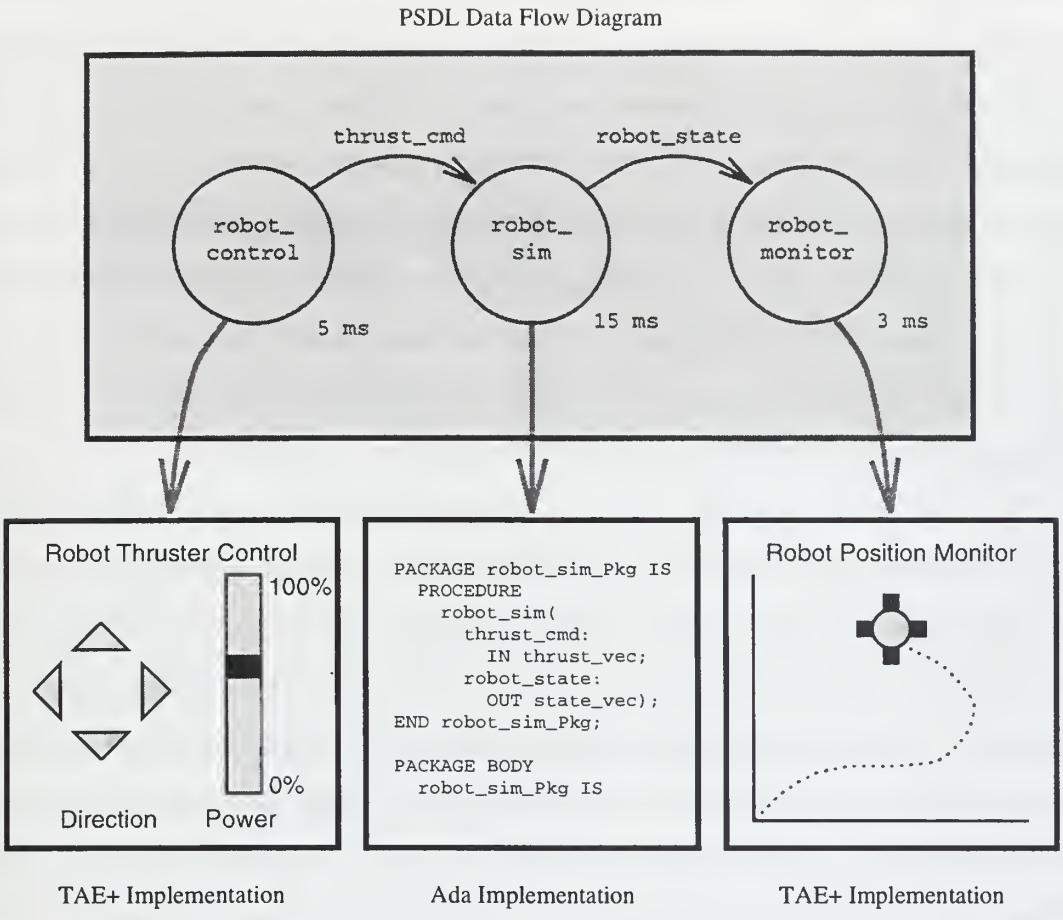


Figure 2. PSDL Data Flow Diagram and Implementation

ingredients of a CAPS prototype. The primary CAPS editor is the PSDL Editor. This editor is unique to CAPS and has been designed specifically for the development of PSDL prototypes. A graphical interface to support the data flow diagram as well as syntax generation and checking facilitate improved user efficiency in prototype development. Other editors are provided to edit the Ada packages and the TAE graphical interfaces which realize the prototype.

The Software Base subsystem provides CAPS with a repository of reusable prototype components. Facilities are provided to browse as well as query the component repository. Queries can be performed using the PSDL component specification.

[Ref. 6]

The Execution Support subsystem is one of the key features of CAPS which allows for rapid prototyping of real-time systems. The Execution Support subsystem provides for the automatic generation of "supervisory" code based on the PSDL specification of the prototype. The "supervisory" code provides scheduling of time critical and non-time critical operators, implements buffered communication paths (e.g., data streams) with applicable initial values, implements the control constraints specified in the prototype, and provides support for timers and exception handling. All of which is automatically implemented by CAPS.

The Project Control subsystem provides advanced project capabilities. The Evolution Control System facilitates prototype development in a team environment. The CAPS Merger provides an automated tool for change-merging of PSDL components. [Ref. 6]

CAPS has been an ongoing research area at the Naval Postgraduate School (NPS) for nearly ten years. During this time, CAPS has been used to develop a variety of student projects and theses, including an autopilot system, a cruise missile guidance system, and a Communications, Command and Control Information warfare (C3I) system. The current version of CAPS is Release 1. [Ref. 7]

B. CAPS RELEASE 1 PSDL EDITOR

The PSDL Editor provided in CAPS Release 1 facilitated the full specification of a prototype in PSDL (refer to Appendix A for the definition of the PSDL grammar). Departing from typical editors, the PSDL Editor provided facilities to view and edit the PSDL augmented data flow diagram in its most natural form, a graph (this functionality will be referred to as the "graph editor" in this document). However, the graph editor was limited in its capabilities to fully specify a prototype. The graph editor provided for the entry of the PSDL data flow diagram along with associated labels (i.e., operator and data stream names). In order to maintain a sim-

ple abstraction of the processing to be performed, the PSDL properties[1] displayed on the augmented data flow diagram were limited to those deemed most critical. For operators, the MET was supported. For data streams, the graph editor provided for the selection between a data stream and a state stream as well as for any stream latency. The remainder of the PSDL specification was supported through the use of a text editor.

Once again, the CAPS designers departed from typical editors with the inclusion of a syntax-directed editor to facilitate the required text editing. A syntax-directed editor is a language-based editor. With knowledge of a language's syntax, a syntax-directed editor is capable of detecting and locating syntax errors. For CAPS, this means that PSDL syntax errors can be corrected in the editor rather than waiting for them to be detected by the Execution Support subsystem. In addition, a syntax-directed editor can provide the user with templates for structures within the language. The syntax-directed editor provided in CAPS Release 1 was implemented using the Synthesizer Generator, a commercial product developed by Thomas Reps and Tim Teitelbaum [Ref. 8].

Provided with a definition of a language's abstract syntax, context-sensitive relationships, display format, concrete input syntax, and transformation rules, the Synthesizer Generator produces a language-based editor. Fundamental to the operation of the Synthesizer Generator is the use of an attribute grammar. An attribute grammar is obtained by the addition of attributes to the nonterminal symbols of a context-free grammar along with a set of attribute equations. As an object is edited with a syntax-directed editor (created by the Synthesizer Generator), the object is represented internally by a derivation tree, which is based on the attribute grammar. It is the derivation tree which is traversed and modified through editing operations. [Ref. 8]

---

[1]PSDL properties will be discussed in Chapter II. For the present, the MET and the Latency are timing requirements of the prototype. A state stream is a special case of a data stream which provides the prototype with memory.

In the CAPS Release 1 PSDL Editor, the graph editor and the syntax-directed editor were merged together to provide an integrated tool to facilitate the development of prototypes. The PSDL Editor was implemented by executing the syntax-directed editor and the graph editor in separate processes. The syntax-directed editor was the parent process. Two copies of the graph editor were executed, each in its own process. Upon startup of the syntax-directed editor, the first copy of the graph editor was created as a graph viewer. As the PSDL prototype was traversed in the syntax-directed editor, the graph viewer displayed the applicable data flow diagram. In order to edit the data flow diagram, a second copy of the graph editor was created as a graph editor.

The implementation of the CAPS Release 1 PSDL Editor resulted in the creation of an artificial boundary between the data flow diagram and the associated PSDL properties. This artificial boundary resulted in a two step process of prototype development for the typical CAPS user. In this process, the user would create the data flow diagram of the prototype using the graph editor (reference Figure 3[2]). When the data flow diagram was reasonably complete, the user would return to the syntax-directed editor (reference Figure 4), where the associated properties would be entered.

The separation between the syntax-directed editor and the graph editor fostered a single mind-set of operation in which all of the data flow diagram was entered prior to the use of the syntax-directed editor to enter the associated properties. State streams presented a large potential for error. If a user specified a state stream in the data flow diagram through the graph editor, the state stream was not implemented until the user again specified the state stream in the syntax-directed editor.

The templates available in the syntax-directed editor provided assistance to the user in developing a syntactically correct prototype. At each stage of a prototype's development, the syntax-directed editor was capable of prompting the user for the

[2]This prototype is taken from the `avionics_example` found in Appendix B.

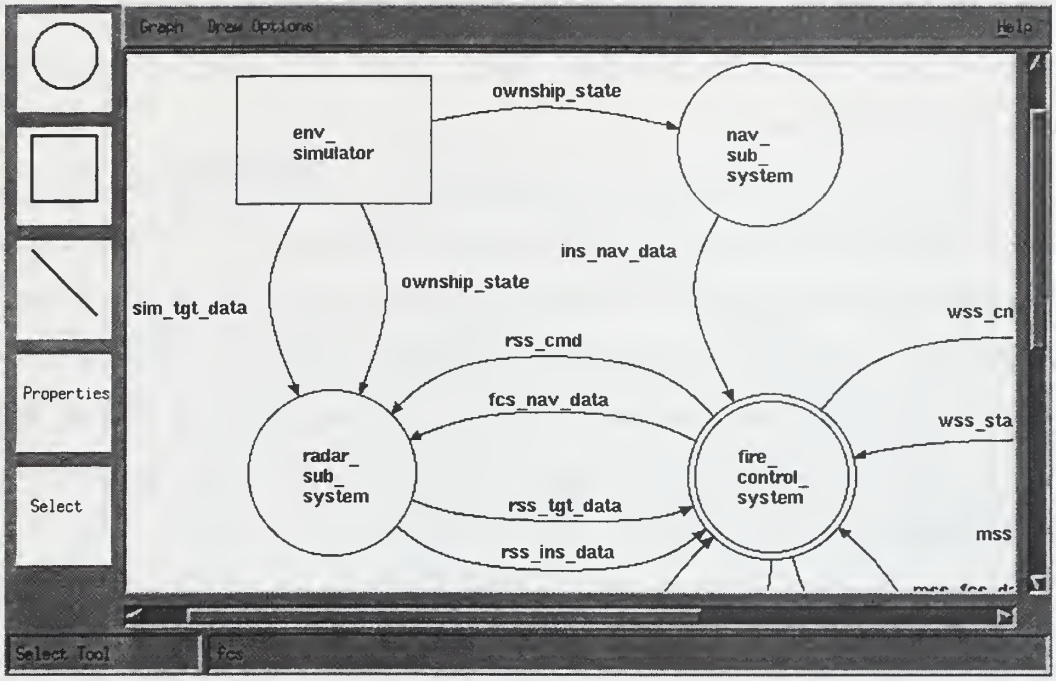


Figure 3. CAPS Release 1 Graph Editor

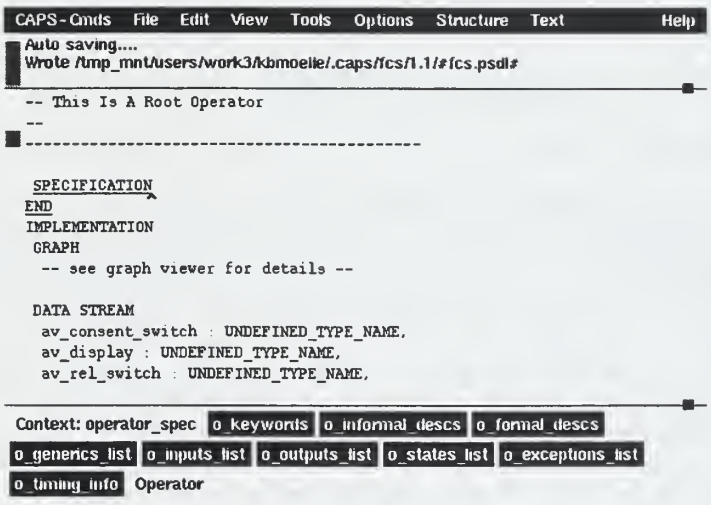


Figure 4. CAPS Release 1 Syntax-Directed Editor

legal alternatives within the PSDL grammar (refer to the bottom of Figure 4). In addition, the knowledge of templates enabled the syntax-directed editor to perform operations on entire structures, made available to the user from the `Structure` menu (refer to the top of Figure 4).

However, due to differences between the attribute grammar programmed into the syntax-directed editor and the PSDL grammar definition (refer to Appendix A), the traversal of the derivation tree maintained by the syntax-directed editor was not always intuitive to the user. In some cases, additional nodes were inserted into the derivation tree which confused the user. In order to become proficient with the PSDL Editor, the user was required to become familiar with the unique features of the attribute grammar.

A typical example is the editing of a `type_declaration` within a `data stream` structure. Figure 5 depicts a graphical representation of the `type_declaration` structure from the PSDL grammar definition (starting with line 25 of Appendix A). Figure 6 depicts how the same structure would be represented in the derivation tree of the syntax-directed editor, based on the attribute grammar. The derivation tree provided additional nodes to represent the set of types which were allowed `INTEGER`, `REAL`, `BOOLEAN`, and `User_Defined`. However, the definition of the attribute grammar was not intuitive to the typical CAPS user. Figure 7 depicts four snapshots of the syntax-directed editor in an example where the user wishes to modify the type used for the data stream `av_consent_switch`. Initially, the data stream is of type `switch_position`, a `User_Defined` type. The required type for the data stream is `BOOLEAN`. The user selects the type name, depicted by the underlined `switch_position` in the first snapshot. The user deletes this entry and expects to be presented with the `decl_type_name` options. However, syntax-directed editor is still located at the `id` node below `DTypeUserDefined` within the derivation tree (refer to the second snapshot). The user is required to ascend to the parent (i.e., `decl_type_name`) from the `Structure` menu and cut off the existing branch of the

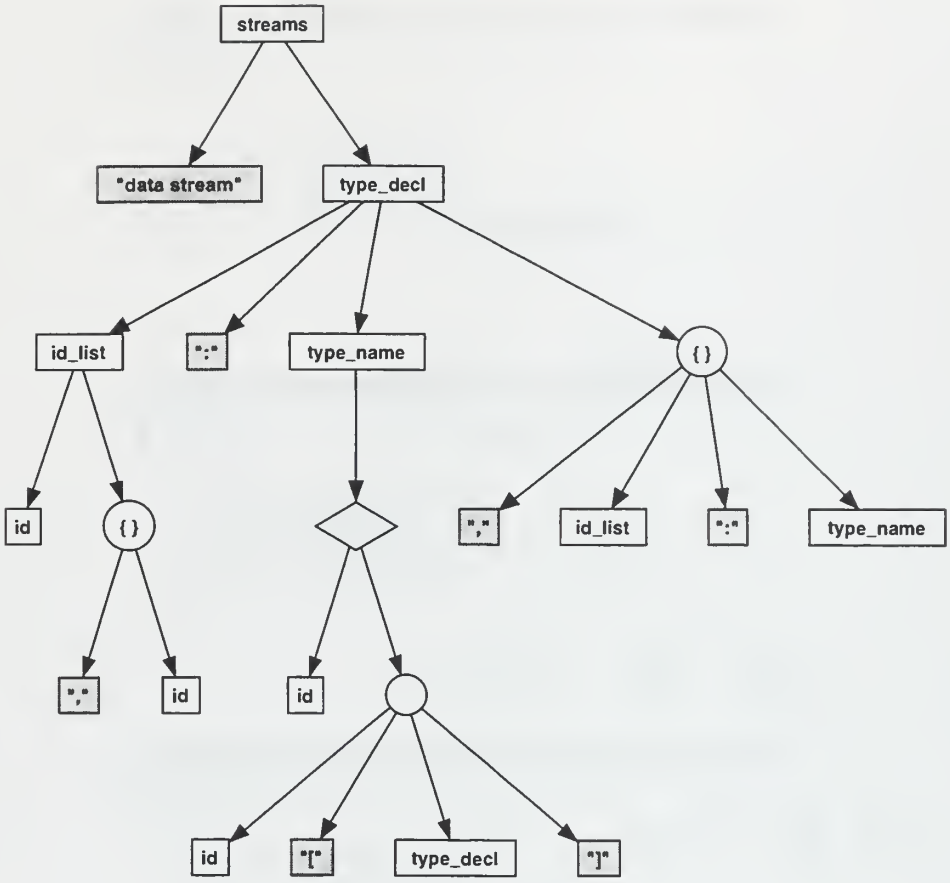


Figure 5. PSDL Type Declaration

derivation tree from the `Edit` menu (depicted in the third snapshot). The syntax-directed editor is now positioned within the derivation tree to prompt for the `BOOLEAN` option, which is selected and depicted in the fourth snapshot.

Additionally, the syntax-directed editor's attribute grammar stipulates an order in which PSDL structures can be arranged. In some cases, this order is defined in the PSDL grammar. In other cases, it is solely defined by the syntax-directed editor. Regardless of the PSDL grammar, the syntax-directed editor requires a specific ordering of PSDL structures. The syntax-directed editor provides the user assistance

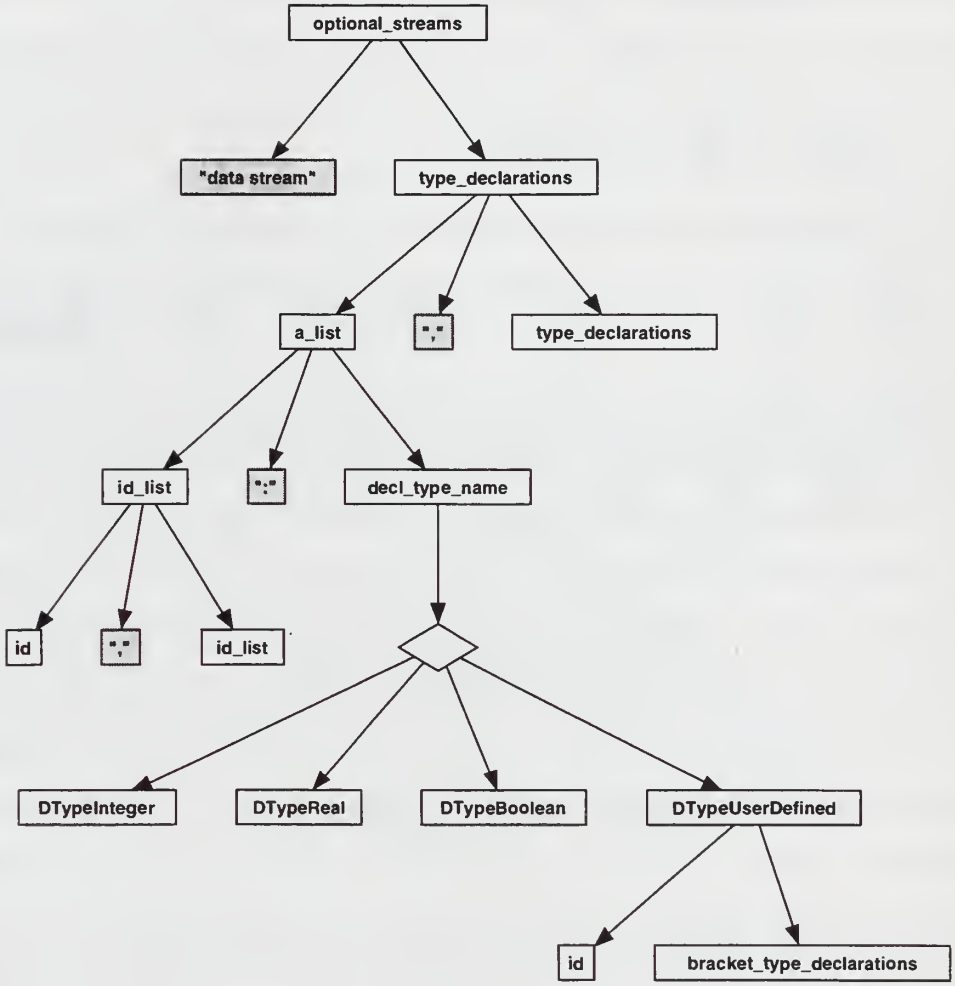


Figure 6. Derivation Tree Type Declaration

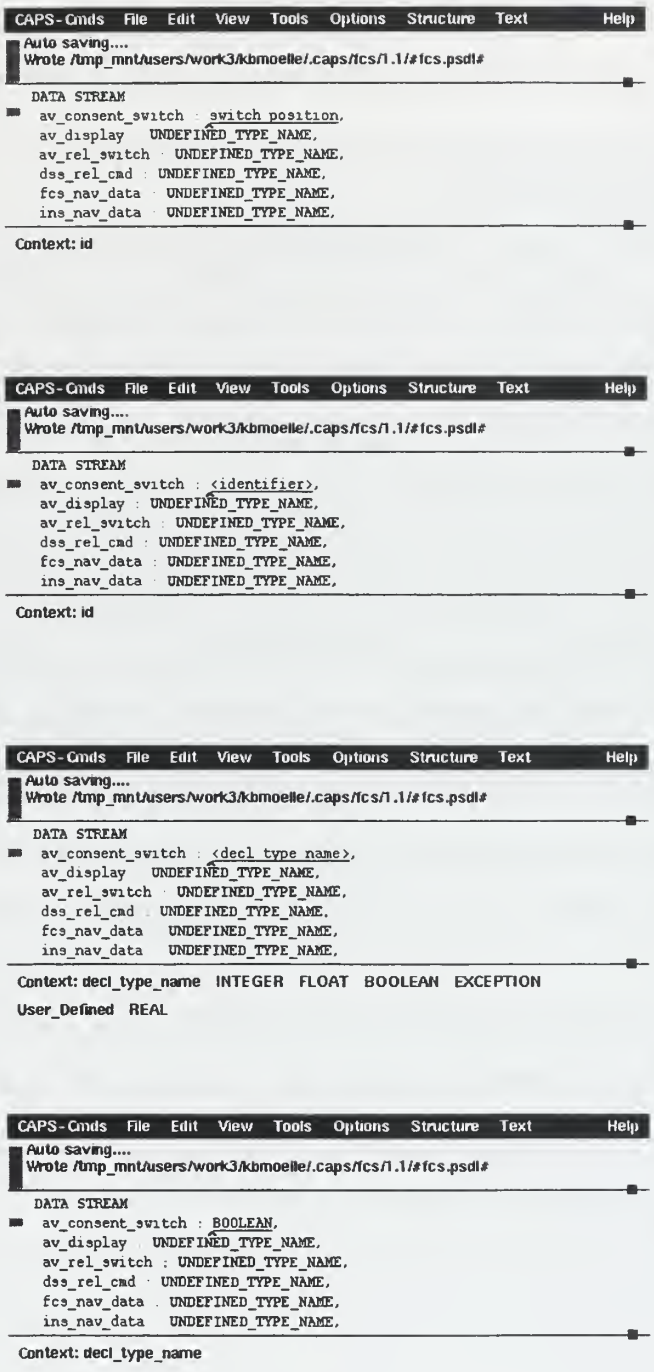


Figure 7. Syntax-Directed Editor: Derivation Tree Traversal

by prompting for all valid constructs at the cursor location. However, the user is required to either memorize the order stipulated by the attribute grammar or to search through the prototype in order to identify the location for the desired construct.

C. RESEARCH GOAL

While the PSDL Editor provided advanced features for specifying the requirements of a prototype in PSDL, it also introduced artificial boundaries and linear constraints that impacted a users productivity. The goal of this research is to develop the next generation of the PSDL Editor in an attempt to overcome these impediments.

As this research was envisioned, the development of the next generation of the PSDL Editor was divided between the graph editor and a background checker driver, written in Ada. This research was centered about the graph editor. This project was implemented as an evolutionary task. The CAPS Release 1 graph editor was used as the baseline from which this research was initiated. Additional work on the graph editor was accomplished partly through the class project for NPS CS4520 (AY96Q4).

II. PROTOTYPE SYSTEM DESCRIPTION LANGUAGE (PSDL)

The primary reference for the Prototype System Description Language is the paper "*A Prototyping Language for Real-Time Software*" by Luqi, Berzins, and Yeh [Ref. 3]; which describes the semantics of PSDL. Refinements to the PSDL semantics are outlined in Professor Luqi's class notes from the Naval Postgraduate School course CS4920: Computer Aided Prototyping Systems (AY96Q3) [Ref. 9]. These two sources, along with the PSDL syntax provided as Appendix A, define PSDL.

This chapter provides an introduction to PSDL as background information for a discussion of the PSDL Editor. Here PSDL is described within the context of a CAPS tool set implementation. This serves two purposes. First, a background in PSDL is necessary in order to understand the required operation of the PSDL Editor. CAPS provides a language-sensitive editor. The PSDL Editor is solely dedicated to CAPS, incorporating many language dependent features designed to simplify the specification of a PSDL prototype. Second, this introduction serves to inform the CAPS user of limitations imposed by a CAPS implementation which deviate from the PSDL semantics.

CAPS provides a set of integrated tools (reference Figure 1) designed to facilitate the development of executable prototypes using PSDL. Limitations within the tool set have resulted in a CAPS implementation that does not fully comply with the PSDL semantics. With each new release of CAPS, an attempt is made to remove these limitations. This research effort comes between CAPS implementations. Currently, CAPS is at Release 1. Release 2 is planned, but not yet fully defined for all CAPS subsystems. Release 2 does contain PSDL syntax changes. These changes are reflected in this introduction as well as in the next release of the PSDL Editor. However, any changes which may be incorporated into the Execution Support subsystem tools (excluding those changes needed to support the new PSDL syntax) are unde-

fined at this time, and hence are not addressed here. Thus, the PSDL configuration which is addressed here is the CAPS Release 1 implementation with the addition of the CAPS Release 2 updated syntax. As a result of the syntax changes which will make up CAPS Release 2, prototypes developed with this PSDL Editor will not be compatible with CAPS Release 1.

Since CAPS users are presented with an implementation which does not fully support the PSDL semantics, users could potentially design a prototype which utilizes a limitation. Limitations with an implementation require special consideration by the prototype developer and may require the designer to over-constrain the prototype in order to work around a limitation. For example, within the PSDL semantics, stream names are local. However, the CAPS Release 1 implementation operates as if stream names are global. In order to maintain the semantics of PSDL, a prototype designer must over-constrain the prototype by forcing unique stream names in order to maintain local streams.

Thus, concessions must be made in the prototype design to make up for limitations in the support provided by the tools. The concessions provided here are in full compliance with the PSDL syntax. They simply over-constrain the design in order to work around a limitation. Prototypes which adhere to these concessions should be compliant with future releases of CAPS. Prototypes which do not adhere to the correct semantics of PSDL will most likely "break" in future releases.

A. PSDL PROTOTYPE

Figure 8 depicts the partial taxonomy of a PSDL prototype. A PSDL prototype consists of a set of components. PSDL provides two kinds of components: abstract state machines and abstract data types. An operator is an instance of an abstract state machine and can be introduced into the prototype by either an abstract state machine component or an operator of an abstract data type component. In PSDL, the operator is the basic computational entity [Ref. 10].

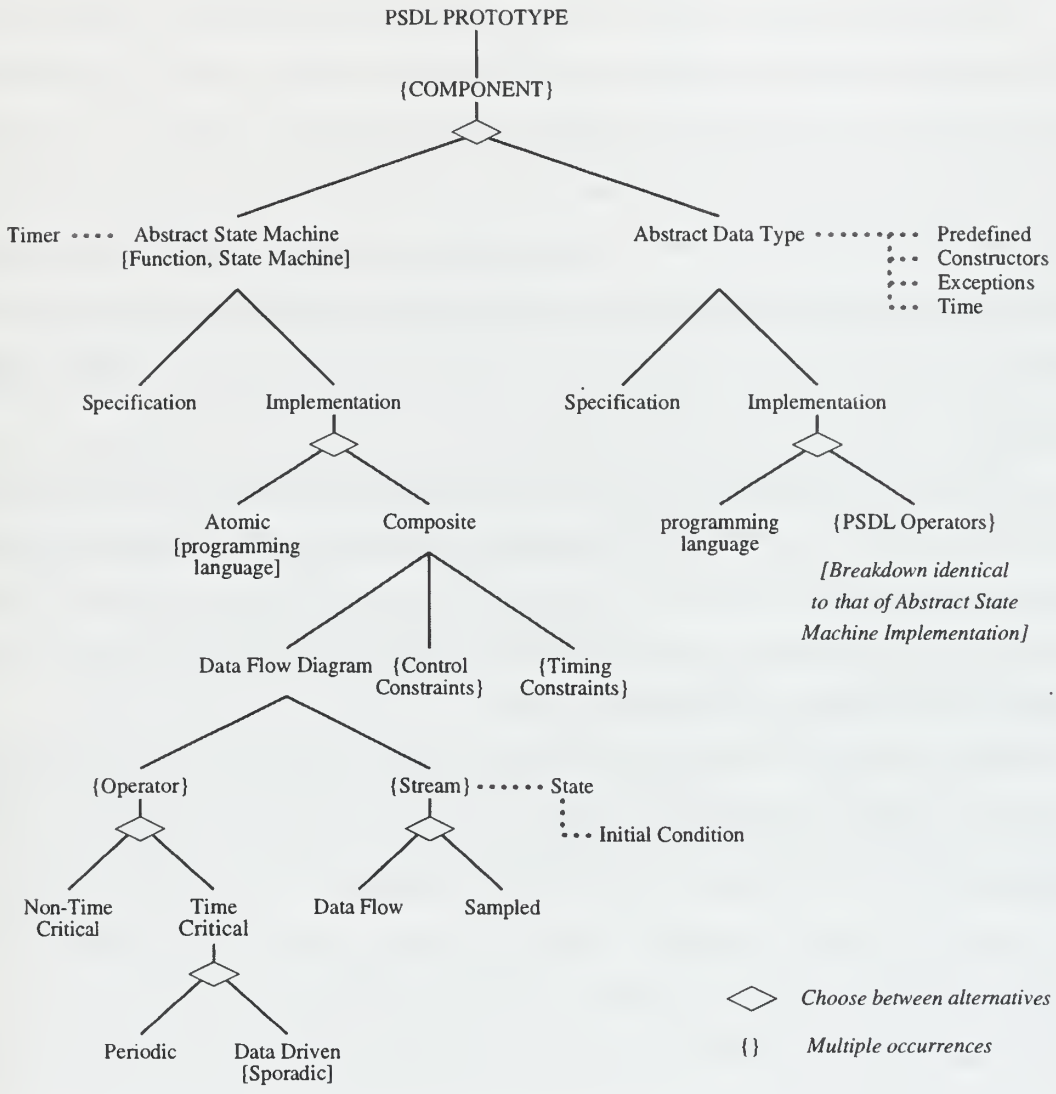


Figure 8. PSDL Taxonomy

A prototype is decomposed into a network of operators which communicate through data streams. Data streams carry objects of a fixed data type. Data stream objects can be either predefined data types or types defined by abstract data type components.

The PSDL syntax provides for the description of this network as a data flow diagram in which the vertices represent operators and the edges represent data streams. By itself, the data flow diagram is not sufficient to support the definition of a real-time system. PSDL augments the data flow diagram with control and timing constraints. The augmented data flow diagram together with a set of precedence rules provides sufficient information for the CAPS Execution Support subsystem to automatically generate the "supervisory" code used to control the execution and communications of operators.

While the operator is the PSDL computational entity, the PSDL syntax does not provide for the complete implementation of all operators. A general purpose programming language is required to implement some operators[3]. Currently, CAPS supports the programming languages Ada and TAE to implement operators. CAPS generates Ada code for "supervisory" control.

The "supervisory" code generated by the CAPS Execution Support subsystem controls the execution of PSDL operators. As can be seen from Figure 8, PSDL contains both time critical and non-time critical operators. Time critical operators are controlled by a static schedule. Non-time critical operators are controlled by a dynamic schedule. The static schedule is executed as a high priority Ada task. The dynamic schedule is executed as a lower priority Ada task that runs whenever the static schedule idles. [Ref. 9]

---

[3]Operators which are implemented using a general purpose programming language are called atomic operators.

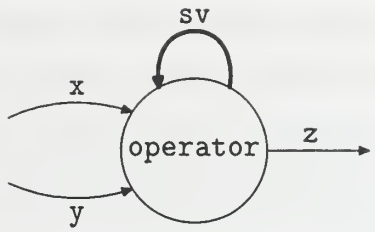


Figure 9. State Machine example

1. Component Structure

Every PSDL component is made up of two parts: a specification and an implementation. The specification defines the component's interface and provides for the documentation of behavior and requirements tracing. The implementation can be performed using a PSDL supported programming language (for atomic operators) or further defined in PSDL (for composite operators).

2. Abstract State Machines

A PSDL operator is a state machine. A state machine contains a finite number of inputs, outputs, and state variables; each of which are represented as a data stream. Figure 9 provides an example of a state machine with two input streams (x and y), one output stream (z), and one state variable[4] (sv). A function is a state machine with no state variables.

When the state machine is executed, it reads one data object from each of the input streams. The output values of the state machine depend solely upon the current values of the input objects that were read and the current value of the state variables. At most, one data object is written to each output stream. [Ref. 3]

An operator that is implemented using a PSDL supported programming lan-

[4]In this example, state variables are depicted with bold lines. This is the convention used by the graph editor.

guage is referred to as atomic. In this case, the operator's implementation specifies the programming language as well as the module name used to implement the operator. Operators that are not atomic are composite. A composite operator is itself decomposed into a network of operators; communicating through data streams. This establishes a parent-child relationship between operators. The composite operator being the parent and the operators contained in the decomposed network being the children.

Composite operators provide for a hierarchical decomposition of a prototype. At the top most level, the prototype consists of a single operator, referred to as the root operator[5]. Children of the root operator can either be implemented as atomic operators or as composite operators. At the lowest level, all operators are implemented as atomic operators.

PSDL provides timers as predefined abstract state machines. A timer behaves similar to a stopwatch. A timer is modeled as an elapsed time value and a run switch. As long as the run switch is on, the elapsed time value is incremented. Timers have four operations: start, stop, reset, and read. Start turns on the run switch. Stop turns off the run switch. Reset turns off the run switch and sets the elapsed time value to zero. Read returns the current value of the elapsed time. [Ref. 3]

Timers are declared in the implementation section of a composite operator. Timers differ from operators in that they do not appear in the data flow diagram. Timer values (i.e, the result of a read operator) are accessed by referring to the timer by name within the control constraints of an operator. PSDL semantics provide for the insertion of a timer value into a data stream [Ref. 3]. Currently, this feature is not supported in the CAPS tools. The prototype designer should limit the use of timers to operator constraints.

---

[5]The PSDL Editor only supports the root operator being implemented as a composite operator.

3. Abstract Data Types

The abstract data type component introduces a new type name for a user defined type. An abstract data type defines a set of values and a set of operations on that value set [Ref. 11]. User defined data types are provided in addition to the set of predefined data types: `boolean`, `character`, `string`, `integer`, `real`, and `exception`. All PSDL abstract data types are immutable. An immutable type has a fixed set of values and the properties of an instance of a type cannot be changed [Ref. 11]. PSDL has no mutable types or global variables[6] in order to prevent coupling problems between operators [Ref. 3].

The set of operations defined by an abstract data type component are implemented as PSDL operators. Similar to operators defined by an abstract state machine component, an operator defined in an abstract data type component can be implemented either in a PSDL supported programming language or as a PSDL augmented data flow diagram. Since an immutable type has no memory (i.e., no state variables) [Ref. 11], it is a good design principle to limit abstract data type operators to functions and avoid state machines and timers.

Data streams carry objects of an abstract data type, the data stream type being assigned in a PSDL type declaration construct. Abstract data type operators can appear within a data flow diagram as an operator by providing the type name and the operator name separated by a '.' (e.g., `stack.push` where `stack` is the type name and `push` is the operator name).

Unusual behavior of an operator can be flagged through the use of an exception. PSDL facilitates this through the use of the built-in abstract data type of `exception`. Exceptions are identified by name. PSDL provides for the raising of an exception of a given name, detecting the presence of an exception with a given name,

[6]The PSDL semantics specify that all streams are local. However, as currently implemented in CAPS Release 1, streams are global, based on the stream name. Thus, to be consistent with PSDL semantics, streams that are not local should have unique names. This is a concession made in the design of the prototype to work around a limitation in the CAPS tool set implementation.

and determining if no exception was raised (i.e., `normal`).

The PSDL syntax provides for a time literal (reference production rule 36 of Appendix A) which consists of an integer and an associated time unit (i.e., microseconds, milliseconds, seconds, minutes, or hours). CAPS Release 1 does not support any time resolution less than one millisecond. Thus, the CAPS Execution Support subsystem converts all time literals to a natural number of milliseconds. Any time literal less than a millisecond is converted to one millisecond. The PSDL does not provide a predefined abstract data type which can be used to support time literals.

B. DATA FLOW DIAGRAM

At the top level, a prototype consists of an operator, whose implementation is provided by an augmented data flow diagram. After the initial operator (i.e., the prototype level), the designer may choose to implement the behavior of an operator with a network of simpler operators or with an atomic operator. This process of decomposing complex operators into simpler composite operators is continued until all remaining operators are atomic. The resulting structure is a hierarchical tree, in which the root is the prototype operator, all internal nodes are implemented as composite operators, and all leaf nodes are implemented as atomic operators.

An augmented data flow diagram consists of a data flow diagram depicting a network of operators (communicating through data streams) as well as control and timing constraints on the network. A sample prototype is depicted in Figure 10, in which two data flow diagrams are included. The root operator is implemented as a composite operator, and hence has an associated data flow diagram. The operator `b` is also a composite operator and contains an associated data flow diagram.

A data flow diagram can accept inputs from an external producer and produce outputs which are used by an external consumer. The inputs and outputs of a composite operator are declared in the composite operator's specification section. These inputs and outputs correspond to the inputs and outputs depicted in the parent data

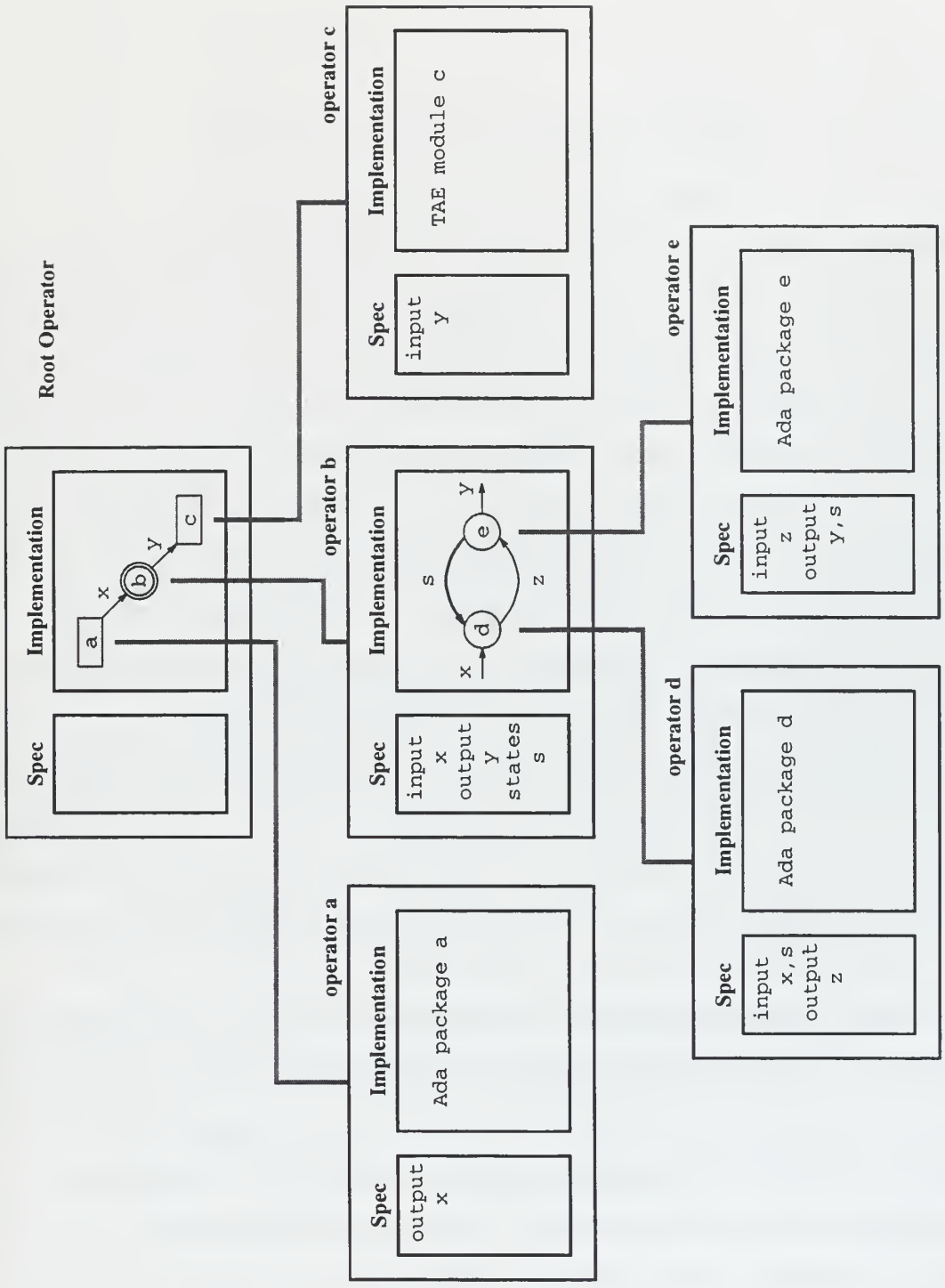


Figure 10. Sample PSDL Decomposition

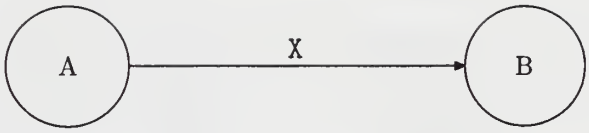


Figure 11. Operator Precedence Relationship

flow diagram. This can be seen in Figure 10. Within the operator `b` component, the specification declares `x` to be an input and `y` to be an output. These inputs and outputs correspond to the streams into and out of the subject operator (i.e., `b`) depicted in the parent data flow diagram (i.e., the root operator). The root operator corresponds to a closed system, in which there are no external inputs or outputs.

While the PSDL syntax supports a textual description of the augmented data flow diagram, the PSDL Editor provides a graphical interface to the user for creating, maintaining and browsing the data flow diagram. Within the graphical interface, an operator is represented as a bubble and a data stream is represented as a directed line segment from the producer operator to the consumer operator.

A precedence relationship establishes a partial ordering of the execution schedule based on the data stream paths between operators. Figure 11 depicts two operators, `A` and `B`, which share a data stream, `X`. The execution of `B` requires the availability of data from `X`. Hence, operator `A` must be executed prior to the execution of operator `B`.

Figure 12 depicts the same state machine with a feedback loop added, data stream `FB`. In this situation, each operator depends on the output of the other operator. CAPS requires that every cycle within a data flow diagram be broken with a state stream. That is, one of the data streams on the cycle must be designated a state stream and provided with an initial value. The initial value of the state stream breaks the circular precedence relationship that would otherwise be impossible to schedule. In order to obtain a valid PSDL prototype, the designer must ensure that the data streams (excluding state streams) of every data flow diagram form a directed acyclic

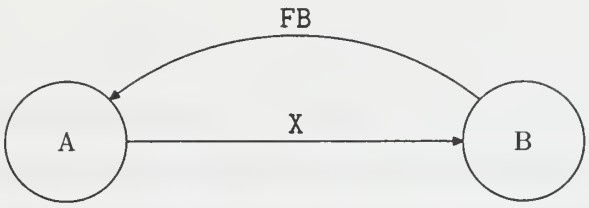


Figure 12. Cyclic Precedence Relationship

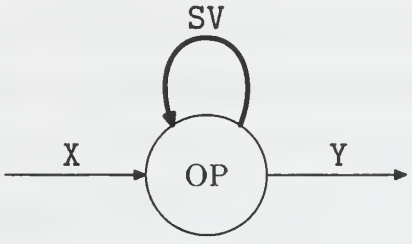


Figure 13. Data Flow Diagram Loop

graph (DAG). Each cycle which is found in the data streams of a data flow diagram must be broken, either by the removal of a data stream or by designating one of the data streams to be a state stream. This rule applies to simple loops as depicted in Figure 13.

1. Operators

In support of real-time prototypes, PSDL provides both time critical and non-time critical operators (reference the taxonomy of an operator depicted in Figure 8). Execution of time critical operators can be triggered either periodically or sporadically (i.e., data driven).

In order for the CAPS Execution Support subsystem to obtain a schedule which executes each operator consistently with the timing constraints of the augmented data flow diagram, a bound must be placed on the execution time of the operators. This bound is referred to as the **maximum execution time** (MET). An operator is time critical if and only if it has been assigned an MET. Otherwise, the

operator is non-time critical. [Ref. 3]

A time critical operator is triggered periodically if it contains a **period** (P) timing constraint. Otherwise, the operator is triggered sporadically, based on the arrival of data (i.e., data driven), and must contain a data trigger control constraint.

Since a PSDL prototype represents a closed system, often it is necessary to include operators which are not considered to be part of the prototype to create the closed system. PSDL facilitates the inclusion of these external systems as terminators. Terminators are operators with an assigned MET of zero[7], and thus the time required to execute the terminator is not counted against the prototype execution time. The CAPS maintains a simulated real-time clock. During the execution of a terminator, the simulated real-time clock is turned off. Terminators are represented in the PSDL Editor by a rectangular bubble within the data flow diagram. Operators with a non-zero MET (including those operators with an undefined MET) are represented in the PSDL Editor by a circular bubble.

2. Streams

Streams are used to communicate data objects of a fixed data type from a set of one or more producer operators to a set of one or more consumer operators [Ref. 9][8]. While the PSDL syntax and the PSDL Editor represent a stream as a link from one producer to one consumer, multiple producers and multiple consumers are supported by matching stream names (i.e., identifiers) within the stream scope.

PSDL provides two types of data streams: data flow streams and sampled streams. A data flow stream guarantees that no data object is lost or replicated. The data flow stream behaves like a first-in-first-out (FIFO) queue with a length of one for each consumer. A sampled stream behaves like a single memory cell which contains one data object for each consumer. The most recent data value is obtained each time

[7]Since all terminators have an MET of zero, they are considered time critical and are placed on the static schedule.

[8]This implementation is a change from that presented in [Ref. 3].

the stream is read. Thus, data objects may be lost if associated with a fast producer or replicated if associated with a slow producer. The type of data stream used is determined by the consuming operator's trigger control constraint. If the consuming operator contains a **triggered by all** control constraint for a set of streams, those streams are data flow streams. Otherwise, the stream is a sampled stream.

The FIFO queue length of one for a data flow stream restricts the relative execution rates between the producer and consumer operators. In order to guarantee that no data object is lost or replicated, the output rate of the producer must not exceed the execution rate of the consumer for all data flow streams. No such restriction is placed on a sampled stream.

3. State Streams

State streams provide a state machine with memory. State streams are also used to schedule data flow diagrams that would otherwise be impossible to schedule due to circular precedence constraints. A state stream is declared in the specification section of the component in which the state stream first appears in the data flow diagram. In Figure 10, the state stream `s` first appears in the data flow diagram of operator `b`. The state stream declaration appears in the specification of operator `b` as well. Within the components of operators `e` and `d`, the producer and consumer of `s` respectively, `s` only appears within the specification's output and input declarations.

A state stream differs from a data stream in that a state stream must include an initial value. It is the initial value of a state stream which makes it possible for a state stream to break a circular precedence constraint. A state stream is also required when connecting time critical and non-time critical operators [Ref. 9].

Note that, under CAPS Release 1, all state streams are implemented as sampled streams regardless of the trigger constraint. Data flow state streams are not provided.

Table I. PSDL Constraints

Constraint	Operator		
	Non-Time Critical	Time Critical	
		Periodic	Sporadic
MET	Undefined	Required	Required
Period		Required	
Finish Within		default: Period	
MRT			Required[a]
MCP			default: MRT-MET
Data Trigger	Optional	Optional	Required
Conditional Exec	Optional	Optional	Optional
Output Guard	Optional	Optional	Optional
Exceptions	Optional	Optional	Optional
Timers	Optional	Optional	Optional

Constraint	Operator	
	Terminator	Operator
MET	0	Otherwise

Constraint	Stream	
	Data Flow	Sampled
Data Trigger	by all[b]	Otherwise

[a]If MCP is set instead, MRT defaults to MCP+MET.

[b]Under CAPS Release 1, all state streams are implemented as Sampled Streams.

4. Constraints

Constraints augment the data flow diagram in order to control the execution of operators, generation of output, processing of exceptions, and timers. Constraints also determine the implementation of data streams. Table I lists the constraints required to obtain a specified functionality of operators and streams as well as those constraints which are optional.

Execution guards provide for the conditional execution of an operator. Execution guards are provided by the constructs:

operator ⟨*op_id*⟩ [**triggered by all** ⟨*id_list*⟩] [**if** ⟨*expression*⟩]
operator ⟨*op_id*⟩ [**triggered by some** ⟨*id_list*⟩] [**if** ⟨*expression*⟩]

Only one of the above constructs is permitted for each operator of a data flow diagram. Each construct has two conditions, a data trigger condition and an if condition. The

two conditions can be used independently or together to provide an execution guard on the operator.

The data trigger condition **triggered by all** is satisfied only if all of the streams listed in the ⟨*id_list*⟩ have new data objects (i.e., has been written to since the last read operation). The data trigger condition **triggered by some** is satisfied if at least one of the streams listed in the ⟨*id_list*⟩ has a new data object. A data trigger condition is required for a sporadic operator. It is an optional execution guard for periodic and non-time critical operators. As can be seen in Table I, the **triggered by all** condition is also used to designate a stream as a data flow stream for the consuming operator.

The second condition within the execution guard is the if condition. Identifiers of the conditional ⟨*expression*⟩ are limited to those of input streams as well as those of visible timers and exceptions.

Output guards provide for the conditional transmission of an operator's output stream. The operator's result is not transmitted over the stream unless the output guard expression is satisfied. One output guard is permitted for each of the operator's output streams. Identifiers of the output guard expression are limited to those of the operator's input and output streams as well as those of visible timers and exceptions.

Exception guards provide for the conditional raising of an exception. Multiple exception guards are permitted, one for each exception to be raised. The resulting exception is transmitted on all output streams of type exception leaving the operator, subject to any output guard constraints defined for those streams. Identifiers of the exception guard conditional expression are limited to those of the operator's input and output streams as well as those of visible timers and exceptions. If the exception guard conditional expression is missing, it is assumed to be true.

Timer control constraints provide for the conditional control of a timer. Conditional control is provided to start, stop, and reset a timer. Identifiers of the timer control conditional expression are limited to those of the operator's input and out-

Table II. PSDL Timing Constraints; From [Ref. 6]

Periodic Operator	Sporadic Operator
Maximum Execution Time (MET)	Maximum Execution Time (MET)
Period (P)	Minimum Calling Period (MCP)
Finish Within (FW)	Maximum Response Time (MRT)

put streams as well as those of visible timers and exceptions. If the timer control conditional expression is missing, it is assumed to be true.

There are five[9] timing constraint parameters used for a PSDL operator. These parameters are listed in Table II, under the class of time critical operators where they are used, and are defined as follows [Ref. 6]:

Maximum Execution Time (MET) An upper bound on the execution time from start to finish of an operator.

Period (P) The elapsed time between two successive triggerings of a periodic operator at the earliest possible moment.

Finish Within (FW) An upper bound on the elapsed time that may pass between the earliest time that a periodic operator can be triggered and the latest time the operator must produce all output stream values.

Minimum Calling Period (MCP) A lower bound on the amount of time that may pass between successive triggerings of a sporadic operator.

Maximum Response Time (MRT) An upper bound on the amount of time that may elapse from the arrival of data which triggers an operator to the time that all output streams have been produced.

The absence of a MET constraint is used to indicate that the operator is non-time critical. Otherwise, the operator is time critical. Only time critical operators are allowed to have timing constraints [Ref. 9]. Only the MET is displayed by the PSDL Editor on the data flow diagram.

[9]MET appears twice in Table II, under both Periodic and Sporadic operators.

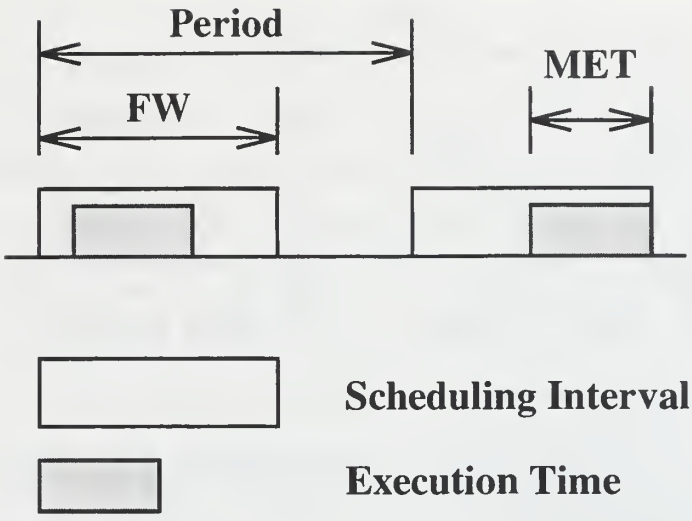


Figure 14. Periodic Timing Constraints; From [Ref. 9]

Periodic operators are scheduled at nearly regular time intervals. Figure 14 depicts the relationship between the timing constraints for a periodic operator. The MET and Period (P) must be assigned for a periodic operator. If a FW constraint is not specified, it is defaulted to P.

Jitter is an upper bound on the time that may elapse between two successive executions of a periodic operator. The worst case of jitter is obtained with the default value of the FW constraint (i.e., FW = P). In such a case, the jitter is given by twice the difference of the P and the MET. Jitter can be reduced to zero by setting the FW constraint equal to the MET.

Sporadic operators are triggered by the arrival of data. Figure 15 depicts the relationship between the timing constraints for a sporadic operator. The MET must be assigned along with either the MCP or the MRT or both.

Limits are placed on the values of MCP and MRT in the absence of a pipelined execution implementation. The MCP must be greater than or equal to the MET. The MRT must be greater than or equal to twice the MET. If the MRT constraint is specified and the MCP constraint is left unspecified, then the MCP defaults to the

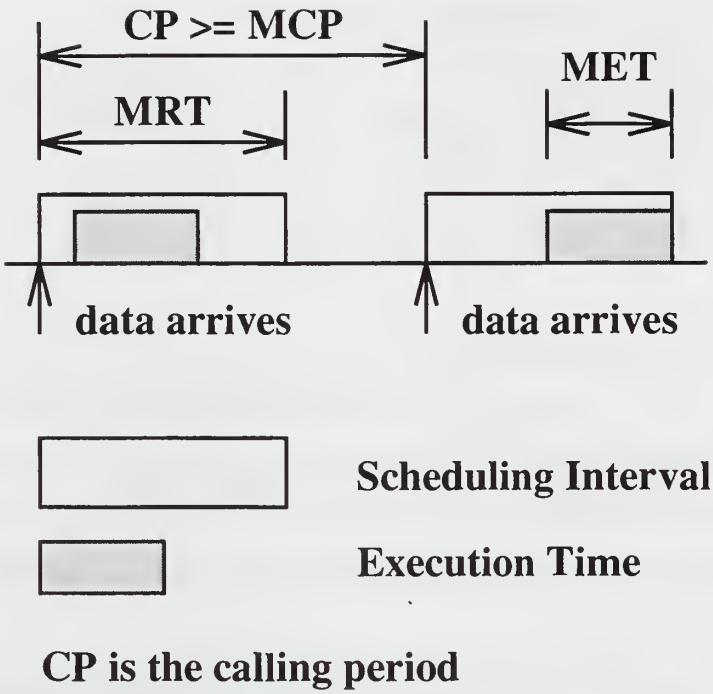


Figure 15. Sporadic Timing Constraints; From [Ref. 9]

difference between the MRT and the MET. If the MCP constraint is specified and not the MRT constraint, then the MRT defaults to the sum of the MCP and the MET.

One additional timing constraint is provided for streams. Latency is the lower bound on the elapsed time from when data is written to a stream by the producer until the time that the data can be read from the stream by the consumer. Latency is also displayed by the PSDL Editor in the data flow diagram.

C. HIERARCHICAL NETWORK

A PSDL prototype is built of PSDL operators, implemented as either data flow diagrams or with a PSDL supported programming language, in a hierarchical manner. The root operator is at the top of the hierarchy, which is always implemented as a data flow diagram. Each of the operators referenced in the data flow diagram is also specified as a PSDL component (either as an abstract state machine or as an

operator of an abstract data type).

The specification of each operator establishes the interface to the operator while the details are hidden in the implementation. For a given operator, the inputs and outputs declared in the specification correlate to the input and output streams of the corresponding operator of the parent's data flow diagram. States depicted in the data flow diagram found in an operator's implementation are also declared in that operator's specification. Exceptions processed in an operator's implementation are also declared in that operator's specification. Timers processed in an operator's implementation are declared in that operator's implementation. The declaration of input and output streams, state streams, and exceptions is provided for with the automatic PSDL code generation of the PSDL Editor. However, the PSDL Editor has no provisions for the automatic generation of the timer declaration. It is the responsibility of the user to properly declare all timers.

The use of a hierarchical structure comes with additional semantic requirements.

1. Root Operator

A PSDL prototype contains a single root operator. Any operator that is not in a hierarchical decomposition of the root operator is considered to be a multiple root operator and represents an error. Use of the PSDL Editor ensures that a multiple root operator error can not occur. A single root operator is provided by the PSDL Editor upon entry. All subsequent operators are introduced as decompositional operators within a hierarchy. Within the PSDL Editor, if a composite operator is deleted, the child operators will be deleted.

CAPS follows the convention that the name[10] of the root operator is the same as that of the file name that contains the prototype. The extension "`.psdl`" is added

[10]In order for CAPS to provide for duplicate operator names in different scopes, the PSDL Editor appends an operator identification number to the operator name. This identification number is not included in the PSDL prototype file name.

to the file name.

While not limited in the PSDL syntax, the PSDL Editor limits the root operator to one which corresponds to a closed system. There are no inputs to, or outputs from, the root operator. If inputs and/or outputs are required, the scope of the prototype should be expanded so that the required inputs and/or outputs are contained within the prototype[11]. The only ⟨*attributes*⟩ (reference production rule 8 in Appendix A) permitted within the specification of the root operator are **generic**, **states**, and **exceptions**.

In addition, the PSDL Editor limits the root operator of the prototype to a PSDL implementation. No provisions are made in the PSDL Editor for a root operator implemented in a programming language.

2. Stream Consistency

Within the hierarchical structure of PSDL, a composite operator is implemented as a data flow diagram of a PSDL component at a lower level. All inputs to the composite operator must be utilized as an external input to at least one of the operators in the decomposed data flow diagram. Likewise, all outputs of the composite operator must be utilized as an external output from at least one of the operators in the decomposed data flow diagram. A similar set of rules require that all external inputs to a decomposed data flow diagram must be inputs to the composite operator and all external outputs to a decomposed data flow diagram must be outputs from the composite operator. [Ref. 3]

External streams in a decomposed data flow diagram inherit the stream's data object type of the the composite operator. In addition, which type of stream (i.e., data flow stream or sample stream) is derived from the trigger constraint of the

[11]User provided inputs as well as output displays to the user are typically handled by an operator (or terminator) contained within the prototype. Such an input/output operator may be implemented in TAE. While user input and/or output may be considered external, the use of an operator (terminator) to capture the requirements and specification of the input and/or output is considered to "close" the prototype system.

consuming operator. For a composite operator, as explained above, an input to a composite operator must also be an input to at least one operator in the decomposed data flow diagram. The trigger constraints of both the composite operator and those of the applicable decomposed operators are used in the derivation of the type. [Ref. 3]

3. Timing

The MET and the MRT of a composite operator must be consistent with the implementation of the operator as a data flow diagram. The MET and the MRT of the implementing data flow diagram must be no larger than that of the composite operator. The Period and the MCP of a composite operator are inherited be the the operators of the implementing data flow diagram.

4. Visibility

The CAPS semantics requires that operator names within a composite operator are to be local to the component. In order to accomplish this, the PSDL Editor supports integer suffixes which are attached to the operator name (reference memo provided in Appendix C). This feature will not be available until CAPS Release 2.

The CAPS semantics provides for all streams to be local. Streams with an identical name out of a common operator are local. Streams with identical names, but out of different operators are not local. However, as implemented in CAPS Release 1, streams are global. That is, any two streams with a common name are visible. In order to maintain the PSDL semantics, all streams that are not desired to be local should have unique names. This is a concession made in the prototype design to workaround an implementation limitation of CAPS Release 1.

When an exception is raised by an operator, the exception is transmitted on all data streams of type exception, regardless of the exception stream label, leaving the operator, subject to the stream output guard. The exception is transmitted only over local exception streams. Thus the exception is not transmitted over exception

streams which are outputs of another operator. At least one exception output stream should be provided for each operator which is capable of generating an exception. [Ref. 9]

A timer is visible within the component in which it is declared. If the component's data flow diagram contains a composite operator, then the timer is visible within the decomposed components.

D. LEXICAL ELEMENTS

Each PSDL component is composed of a sequence of lexical elements. Lexical elements are in turn composed of characters. A lexical element is either an integer literal, a real literal, a string literal, text, an identifier (including keywords), or a delimiter[12].

When a prototype is maintained by the PSDL Editor, the editor will ensure that the prototype is syntactically correct. User input provided through the PSDL Editor's graphical interface are validated as required and are automatically translated into a syntactically correct PSDL prototype. Thus many of the rules presented in this section do not impact the prototype developer since they are maintained by the PSDL Editor. If a developer chooses to maintain a prototype using any other system, it is the responsibility of the developer to adhere to the syntax of PSDL. Very little assistance in the form of error messages is provided to the developer who introduces syntactical errors into a prototype. In most cases, the PSDL Editor will not be able to recover from these syntax errors.

1. Character Set

The PSDL character set is composed of the 95 printable ASCII characters from ASCII '␣' (space) through ASCII '~' (tilde) plus the line feed (ASCII character 10) and horizontal tab (ASCII character 9) characters.

[12]The time literal mentioned earlier is not truly a literal. Rather, it is a grammatical rule which combines an integer literal with a keyword representing the time units.

Right brace, '}', is a special character in that it is included in the PSDL syntax, however, it is not allowed to appear anywhere except the closing of a **description** or an **axioms** structure (reference production rule 45 of Appendix A for the definition of a character and production rules 15 and 16 for the use of '}'). The format effectors vertical tab, carriage return, and form feed can not appear anywhere other than the ⟨*text*⟩ field of the **description** or the **axioms** structures[13].

2. Integer Literals

An integer literal (reference production rule 43 of Appendix A) is an unsigned number. All integer literals are base ten and do not contain any character other than the digits **0** through **9**.[14]

3. Real Literals

A real literal (reference production rule 42 of Appendix A) is an unsigned real number. All real literals are base ten. A real number must contain a decimal value (which may be **0**), a decimal point ('.'), and a fractional value (which may also be **0**). Exponential notation is not allowed.

4. String Literals

A string literal (reference production rule 44 of Appendix A) is any number of characters from the PSDL character set, delimited by quotations ('"'). There are four characters that are not allowed within a string literal: the quotation mark ('"'), the right brace character ('}'), and the format effectors horizontal tab and line feed. PSDL does not specify any limit on string literal length. The empty string is represented as '""'. Strings are case sensitive.

---

[13]The use of these characters is not recommended.

[14]Within the PSDL Editor, integer literals are represented by a C `integer` data type. In the current implementation, this is a 32 bit value. Only the non-negative values can be represented as an integer literal.

5. Text

Text (reference production rule 49 of Appendix A) can only appear within the **description** and the **axioms** structures of PSDL. Within these two structures, text is any number of characters from the PSDL character set, delimitated by braces ('{' and '}'). Hence, the right brace ('}') can not appear within the text. Any other character, including horizontal tab and line feed, are permitted within the text. Text is the only lexical element that is allowed to cross a line separator.

6. Identifiers

An identifier (reference production rule 41 of Appendix A) consists of a letter, followed by any number of letters, digits, or underscore ('_') characters. All characters of an identifier are significant and are case sensitive.[15]

7. Reserved Words

While not addressed in the specification of PSDL [Ref. 3], the implementation of the PSDL Editor utilizes reserved words. A list of these reserved words is provided in Table III. Reserved words can not be used as identifiers within the PSDL prototype. Reserved words differ from identifiers in that reserved words are case insensitive (e.g., **OPERATOR**, **Operator**, and **operator** are all reserved words).

Table IV contains additional PSDL keywords that do not match the syntax of an identifier and hence are not reserved. However, they are still tokens within the PSDL syntax. The symbol '␣' is used to represent a single space. This single space is part of the keyword and can not be altered by removal, adding of additional spaces, or replaced with a horizontal tab.

In addition to the above keywords (Tables III and IV), PSDL defines a set of identifiers that are available to the prototype developer. This set of identifiers and

[15]The PSDL Editor treats identifiers as being case sensitive. Note that Ada identifiers are case insensitive [Ref. 12]. Since CAPS generates control code in Ada it is not advisable to name two identifiers that only differ in case. Moreover, the CAPS Release 2 PSDL Editor will convert all characters of the identifiers to lower-case.

Table III. PSDL Reserved Words

abs	**false**	**min**	**sec**
all	**generic**	**mod**	**some**
and	**graph**	**ms**	**specification**
axioms	**hours**	**not**	**states**
boolean	**if**	**operator**	**timer**
description	**implementation**	**or**	**triggered**
edge	**initially**	**output**	**true**
end	**input**	**period**	**type**
exception	**integer**	**property**	**vertex**
exceptions	**keywords**	**real**	**xor**
external	**microsec**	**rem**	

Table IV. Additional PSDL Keywords

control␣constraints	**maximum␣response␣time**	**start␣timer**
data␣stream	**minimum␣calling␣period**	**stop␣timer**
finish␣within	**required␣by**	**triggered␣by**
maximum␣execution␣time	**reset␣timer**	

their application are provided in Table V.

Table VI contains additional identifiers that are defined by the PSDL Editor for the purpose of describing the data flow graph. These identifiers appear in **PROPERTY** constructs of the PSDL syntax (reference production rules 20 - 23 in Appendix A). The attributes of the data flow diagram that are controlled by these

Table V. Predefined PSDL Identifiers

Identifier	Application
`ADA`	Implementation Language
`TAE`	Implementation Language
`other`	Place holder for other implementation languages
`normal`	Exception
`character`	Predefined data type
`string`	Predefined data type

Table VI. PSDL Editor Identifiers

Identifier	Application
`color`	Integer representing color of Operator
`is_terminator`	`TRUE` if Operator is a Terminator
`label_font`	Integer representing font of Label
`label_x_offset`	Signed integer representing relative y position of Label
`label_y_offset`	Signed integer representing relative x position of Label
`latency_font`	Integer representing font of Latency
`latency_x_offset`	Signed integer representing relative y position of Latency
`latency_y_offset`	Signed integer representing relative x position of Latency
`met_font`	Integer representing font of MET
`met_x_offset`	Signed integer representing relative y position of MET
`met_y_offset`	Signed integer representing relative x position of MET
`radius`	Integer representing radius of Operator
`spline`	List of absolute x, y positions of Splines
`x`	Absolute x location of Operator
`y`	Absolute y location of Operator

identifiers are maintained by the PSDL Editor and are hidden from the user by the PSDL Editor. They will be visible in the PSDL prototype code generated by the PSDL Editor.

8. Delimiters

Delimiters are required to separate adjacent lexical elements. Delimiters include a space character (except when included in a string literal or text), a line feed, or one of the symbols listed in Table VII.

9. Comments

The PSDL syntax does not provide for comments in the sense of comments provided by languages such as Ada. Provisions are made for structured documentation of the prototype. The format and placement of these structures is defined by the PSDL syntax (reference Appendix A). The reserved words **description**, **axioms**, **keywords**, and **required by** are tokens used to introduce documentation into the prototype.

Table VII. PSDL Delimiters

&	,	:	[
(	–	<	]
)	–>	<=	{
*	.	=	|
**	/	>	}
+	/=	>=	

Note that, while **description** and **axioms** constructs facilitate the textual documentation of the prototype, **keywords** and **required by** constructs are restricted to an ⟨*id_list*⟩.

E. PSDL EXECUTION

CAPS provides facilities for executing a PSDL prototype when supported with software components written in an underlying programming language (e.g., Ada) [Ref. 3]. These software components may be obtained from a software base of reusable components or supplied by the prototype developer.

An executable prototype is generated in three steps, available through the Executive Support subsystem of CAPS. Figure 16 depicts the CAPS Release 1 user interface access to these steps[16]. The first two steps (i.e., Translate and Schedule) are used to generate the "supervisory" module which provides for control and communication within the prototype. When Compile is invoked, the "supervisory" module along with the packages used to implement the atomic operators and data types are compiled. The prototype is run by invoking Execute.

Translation actually involves two processes. The first process involves flattening the hierarchical structure of data flow diagrams in an expansion process. The

[16]Note that this version of the PSDL Editor is in support of the PSDL grammar which will be integrated with CAPS Release 2. This PSDL Editor is not compatible with CAPS Release 1. Comments on compatibility can be found in Appendix B. Execution of a PSDL prototype under CAPS Release 1 is covered here since it is likely that the same steps will still be required with CAPS Release 2.

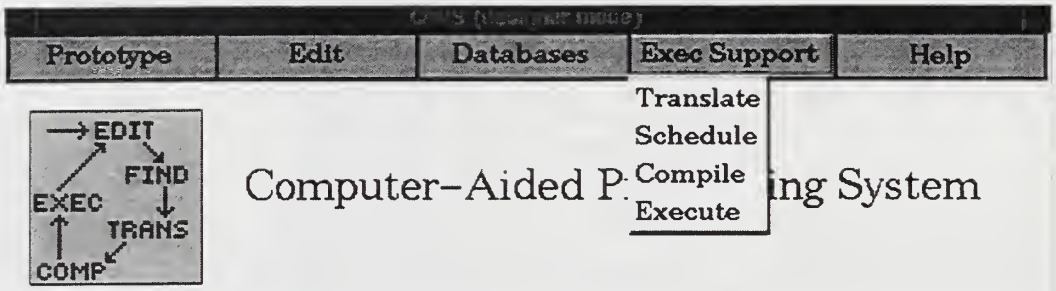


Figure 16. CAPS Release 1 Executive Support

second process is to generate the Ada code required to support the timers, exceptions, stream communication, and control constraints.

The remainder of the "supervisory" module is generated by the scheduler. The scheduler creates a high priority Ada task to control the static schedule (i.e., time critical operators) and a lower priority Ada task to control the dynamic schedule (i.e., non-time critical operators).

All of the generated "supervisory" code is gathered together in one file which is named after the prototype name with an ".a" extension. Which, assuming that the prototype was successfully translated and scheduled, is compiled with the atomic operators and data types. The result is an executable prototype.

III. PSDL SYNTAX/SEMANTIC CONSIDERATIONS

The PSDL Editor is a language sensitive editor with embedded knowledge of PSDL syntax and semantics. The PSDL Editor assists the user by ensuring a syntactically correct prototype, with limited automatic PSDL code generation and limited semantic consistency checking. In the CAPS Release 1 PSDL Editor, this was primarily accomplished by the syntax-directed editor produced by the Synthesizer Generator. The graph editor served two purposes. First, the graph editor was used to present the data flow diagram corresponding to the operator selected in the syntax-directed editor. Second, the graph editor was used to create/maintain the data flow diagram of an operator selected in the syntax-directed editor. Since only the structure, labels, and a few timing parameters of the data flow diagram could be displayed/entered in the graph editor, a simple interface was all that was required to integrate the two segments of the PSDL Editor.

In an attempt to meet the goal of minimizing the artificial boundary produced by implementing the PSDL Editor with both a syntax-directed editor and a graph editor, as well as overcoming the linear constraints resulting from the syntax-directed editor, the user interaction with the syntax-directed editor was to be reduced and the focus placed on the graph editor. As this project is an evolution of the current PSDL Editor, the existing architecture of a syntax/semantics checker and a graph editor was maintained. With the user's focus being placed on the graph editor, an expanded interface was required between the syntax/semantics checker and the graph editor. This change in focus also required a redistribution of the embedded syntax and semantic knowledge between the two editor segments.

This research concentrated on the development of the graph editor. The syntax/semantics checker was implemented as an Ada program, refered to as the background checker, in a parallel development effort by Professor Man-Tak Shing. The

Table VIII. PSDL Interface Structures Summary

Structure	Description	Direction
GRAPH_DESC	Representation of a single operator plus some "global" data.	SDE $\leftrightarrow$ GE
ERROR_MSGS	Semantic errors detected by the syntax-directed editor.	SDE $\rightarrow$ GE
ACTION	Control requests for next action to be taken by the syntax-directed editor.	SDE $\leftarrow$ GE

integration of the the background checker and the graph editor was a shared task. In order to accomplish this parallel development, a new interface was established. Unlike the interface which was utilized in the CAPS Release 1 PSDL Editor, which shared a minimal set of data flow diagram data, this interface shared the complete specification of a single PSDL operator. Additional data was shared to provide "global" prototype information to the graph editor was well as to support control functions. The interface was defined by the file `ge_interface.h` (reference the source listing of `ge_interface.h` found in Appendix D on page 195).

Figure 17 depicts the interfaces of the PSDL Editor. External interfaces to the PSDL Editor provide for the reading/writing of the PSDL code as well as for user interaction with the editor through the Graphical User Interface. Internally, all editor segment communication is performed through the `ge_interface.h` data structures, represented by the dashed box inside the PSDL Editor of Figure 17. The `ge_interface.h` file contains three structures used to organize communications between the background checker and the graph editor. These structures are summarized in Table VIII.

The `GRAPH_DESC` interface accommodates the representation of a single PSDL operator. This is symbolized by the box around operator `d` with an arrow pointing to the `GRAPH_DESC` data structure in Figure 17. The interface does not support the

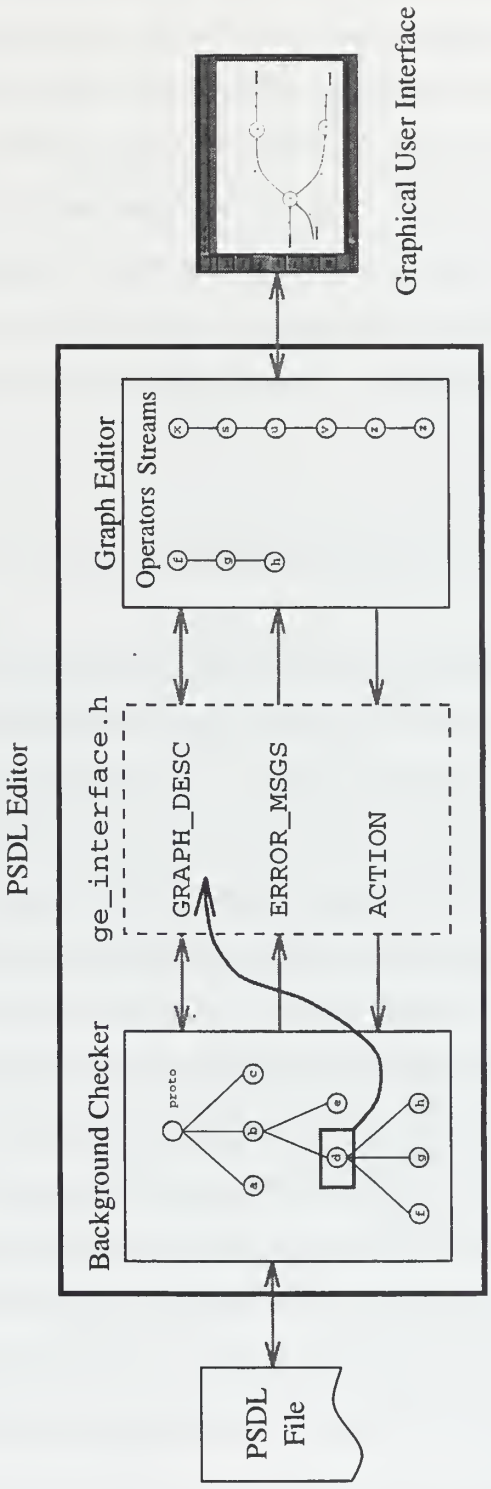


Figure 17. PSDL Editor Interfaces

PSDL representation of the operator[17]. Instead, coming from a PSDL file, the PSDL code is parsed by the background checker and attribute values are stored in the prototype data structure, local to the background checker. PSDL reserved words are removed. The association of an attribute value with a PSDL construct is provided implicitly in the data structure. In the other direction, coming from the Graphical User Interface, attribute values entered by the user are associated with, and stored into, the `GraphObjectList` data structure, local to the graph editor. It is not until the prototype is written to disk by the background checker that the attribute values are mapped into PSDL code.

A. SYNTAX AND SEMANTIC KNOWLEDGE DISTRIBUTION

Since only a representation of the PSDL operator is shared between the background checker and the graph editor, the majority of the embedded syntactical knowledge must reside in the background checker. The background checker is responsible for the input parsing of a PSDL prototype as well as the generation of the PSDL code, which is the output of the PSDL Editor. The syntactical knowledge embedded within the graph editor is limited to that required to validate the user supplied data values associated with a PSDL construct and limited, locally-scoped, semantic validation.

A syntactically correct PSDL prototype is ensured in a two step process. First, all user supplied data values are validated prior to acceptance into the graph editor's data structures. Second, all PSDL code is generated by the background checker, based on the inputs provided by the user from the graph editor.

It is critical that the PSDL Editor generate syntactically correct prototypes. No matter what stage of a prototype's development, the prototype must be syntactically correct. All viewing/editing of a prototype is accomplished from the graph

[17]The `ge_interface.h` accommodates some PSDL code in order to facilitate the editing of selected PSDL constructs which were too complex for the initial implement of the graph editor's user interface. This includes all PSDL types and expressions.

editor. The graph editor does not receive or transmit the PSDL code. The code is encoded into the GRAPH_DESC data structure by the background checker. The background checker is only capable of encoding the PSDL code into the internal data structure representation if it can parse the code. Currently, the background checker does not have any provisions for error correction of a syntactically incorrect prototype. The CAPS PSDL Editor can not be used to edit a syntactically incorrect prototype.

The graph editor is embedded with a limited amount of PSDL semantic knowledge. Semantic consistency is primarily accomplished by limiting the user's options to those that are semantically correct. The graph editor is limited to semantic issues that are local to an operator. The interface between the background checker and the graph editor is limited to a single operator. All global semantic issues are resolved by the background checker.

Note that while a prototype which is not semantically correct can not be translated and executed by the Execution Support tools, it is not necessary that the prototype be semantically correct to be edited with the PSDL Editor. There are potential semantically incorrect prototypes which are not compatible with the PSDL Editor. However, there is no general limitation against semantically incorrect prototypes as there are with syntactically incorrect prototypes.

B. USER SPECIFIED PSDL CONSTRUCTS

It is the graph editor that provides the user interface for PSDL Editor. The graph editor does not understand PSDL syntax. Instead, the graph editor deals with a small collection of data objects from which a PSDL construct is built. It is the data objects provided by the user which are properly formatted with PSDL reserved words by the background checker to produce a PSDL prototype (i.e., PSDL code). The data types which must be supported by the graph editor can be derived from the PSDL grammar (reference Appendix A). Table IX lists the fundamental data objects which must be supported by the graph editor. Production rules provided in Table IX

Table IX. Fundamental PSDL Data Objects

PSDL Data Object	Defining Production Rule
`id`	41
`id list`	11
`operator id`	24
`type name`	10
`integer literal`	43
`text`	49
enumeration	

Table X. Enumeration Values

Enumeration Usage	Enumeration Values, separated by '|'
Operator Type	`Operator | Terminator`
Implementation Language	`ADA | TAE | Others`
Trigger	`unprotected | By Some | By All`
Timing	`Non-Time Critical | Periodic | Sporadic`
Time Literal Units	`microsec | ms | sec | min | hour`
State Stream Selection	`Yes | No`

refer to those specified in Appendix A which define the syntax of the data object.

Enumeration values are provided in Table X. Some of the enumeration values correspond to PSDL reserved words. However, some are used to free the user from implementation details of PSDL. For instance, the operator type (i.e., `Operator | Terminator`) does not correspond to any PSDL construct. The selection of a `Terminator` is encoded by setting the Maximum Execution Time of the operator to zero. In this case, the use of an enumerator, along with semantical knowledge embedded in the graph editor, are used to hide this implementation detail of PSDL from the user.

There were specific PSDL constructs which were deemed too complex to be provided in the initial implementation of the graph editor's user interface. Table XI lists those constructs for which syntactical knowledge was not embedded in the graph

Table XI. Complex PSDL Data Objects

PSDL Data Object	Defining Production Rule
`data type`	3
`interface`	7
`constraint options`	29
`expression`	39
`initial expression list`	32

editor. In order to provide full functionality within this version of the editor, separate syntax checkers are provided for these constructs. The graph editor provides a text window from which these constructs may be viewed/edited[18].

C. HIERARCHICAL STRUCTURE

A PSDL prototype is implemented as a network of operators, which may or may not be connected. Syntactically, a PSDL prototype is simply a set of operators[19]. A given operator consists of a specification and an implementation construct. It is the data flow diagram, contained in the operator's implementation, which describes the prototype network. The prototype network itself is decomposed in a hierarchical manner, described by a tree, in which each operator can be described by its own network of PSDL operators.

Figure 18 depicts the skeleton of a PSDL prototype[20]. The prototype consists of nine operator components. The implementation construct of the root operator, `proto`, identifies the first level of vertices (i.e., operators) and edges (i.e., streams)

[18]Syntactical validation of these constructs is required in order to maintain a syntactically correct prototype. As of this writing, this facility has not yet been provided.

[19]Actually, a PSDL prototype is a set of operators and abstract data types. However, it is only the operators, which includes operators of an abstract data type, which are executed. For this discussion, we will consider operators only.

[20]The code presented here does not conform to the CAPS Release 2 semantics. In order to simplify the example, the unique identifier suffixes have been deleted. The unique identifier suffixes will be introduced later in this chapter.

Figure 18. Sample PSDL Prototype

which constitute the prototype network. Each operator identified in this network is contained in the PSDL prototype, the implementation of which may or may not be described as a network of PSDL operators. Figure 19 provides a corresponding hierarchical representation, along with a simple tree representation, of the PSDL prototype provided in Figure 18.

Note that, while the decomposition of the PSDL network is represented as a tree, the PSDL network itself is not restricted to a tree. In general, the PSDL network can be implemented as a disconnected graph containing cycles. Figure 20 depicts how the prototype tree structure is flattened into the PSDL network, which implements the prototype, `proto`. For each level of the tree, the resulting PSDL network is provided. The resulting network is representative of the outcome of the flattening process used by the Execution Support tools. While the original network of Figure 18, described by operator `proto`, was a simple tree consisting of three vertices, the flattening of the PSDL network resulted in a connected graph of six vertices containing a cycle.

The tree describes the relationship between operators, not the communication paths between operators. In Figure 18, operator `d` has been selected as the current operator of interest. From Figure 19, it can be seen that operator `b` is the parent and that operators `f`, `g`, and `h` are the children. Operator `proto` is the root operator.

Within the background checker, the entire PSDL prototype is maintained. The interfacing data structure, `GRAPH_DESC` (contained in `ge_interface.h`, reference Appendix D, page 195), only facilitates the representation of one operator. The current operator being edited is selected by the user, in a process of navigating through the prototype tree structure. Initially, the root operator is selected by the PSDL Editor as the current operator.

Thus, for the prototype depicted in Figure 18, all nine operators would be maintained simultaneously within the background checker's data structures. Within the graph editor, a single operator, such as the data represented by the shaded rectangle (operator `d`) in Figure 19, would be maintained. Initially, the operator `proto`

Figure 19. Sample PSDL Hierarchy

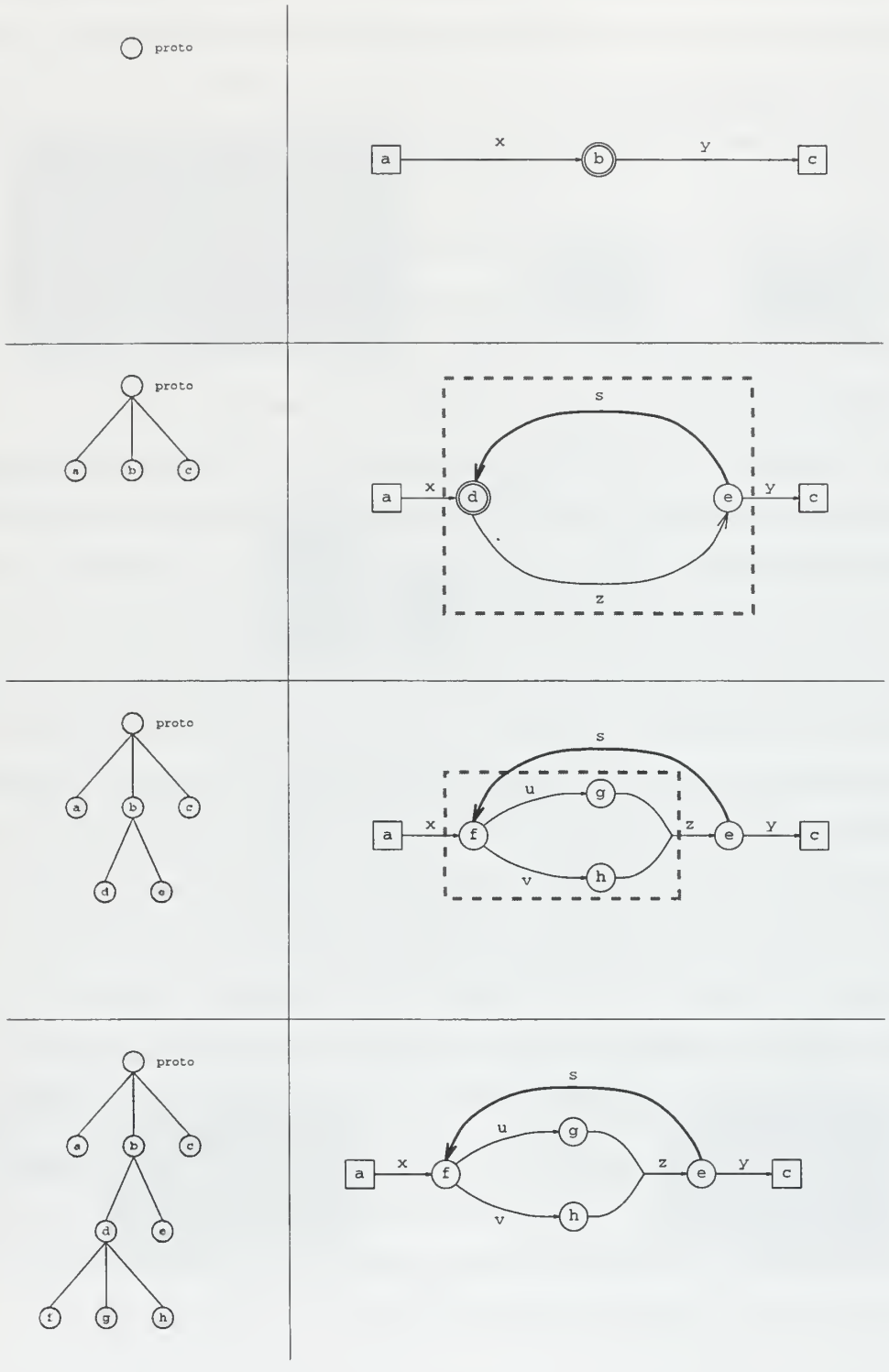


Figure 20. Flattening of the PSDL prototype

would be presented to the user by the graph editor. As the user navigates through the operator tree, the operator represented in the graph editor data structures is replaced with the operator selected by the user. It is the responsibility of the background checker to extract and relay the information for the current operator to the graph editor.

From a graph editor perspective, at most four levels of the prototype tree are relevant at any given time. It is the responsibility of the background checker to orchestrate the data of these four levels with the information contained in the GRAPH_DESC data structure. These four levels are represented in Figure 21, in which the graph editor has no visibility into the shaded regions[21]. The operator of interest is the current operator. This is the operator which is being edited. Both the specification and the implementation constructs of the current operator are available for viewing and editing. One level down from the current operator are the children operators. Portions of the specification construct of the child operator are available for viewing and editing. The input/output constructs of the child's specification are derived from the current operator's data flow diagram. The functionality constructs of the child's specification are also made available through the editing of operator options in the current operator's data flow diagram. Also available from the operator options is the implementation language of the child operator. One level above the current operator is the parent. The parent operator is used by the background checker to derive the current operator's specification construct (i.e., the input and output constructs). The parent operator is not directly visible from the graph editor. The final level which is visible is the root level. The actual root operator is not visible (unless it is the current operator). However, the root level contains global information, which is visible from any level. Specifically, abstract data types are global to the prototype and are visible from any level of the prototype tree.

[21] Note that, while the graph editor may have visibility into a region which is not shaded, the graph editor does not necessarily have the ability to write to an area that is not shaded. Much of this visibility is provided by the background checker providing derived information on the prototype.

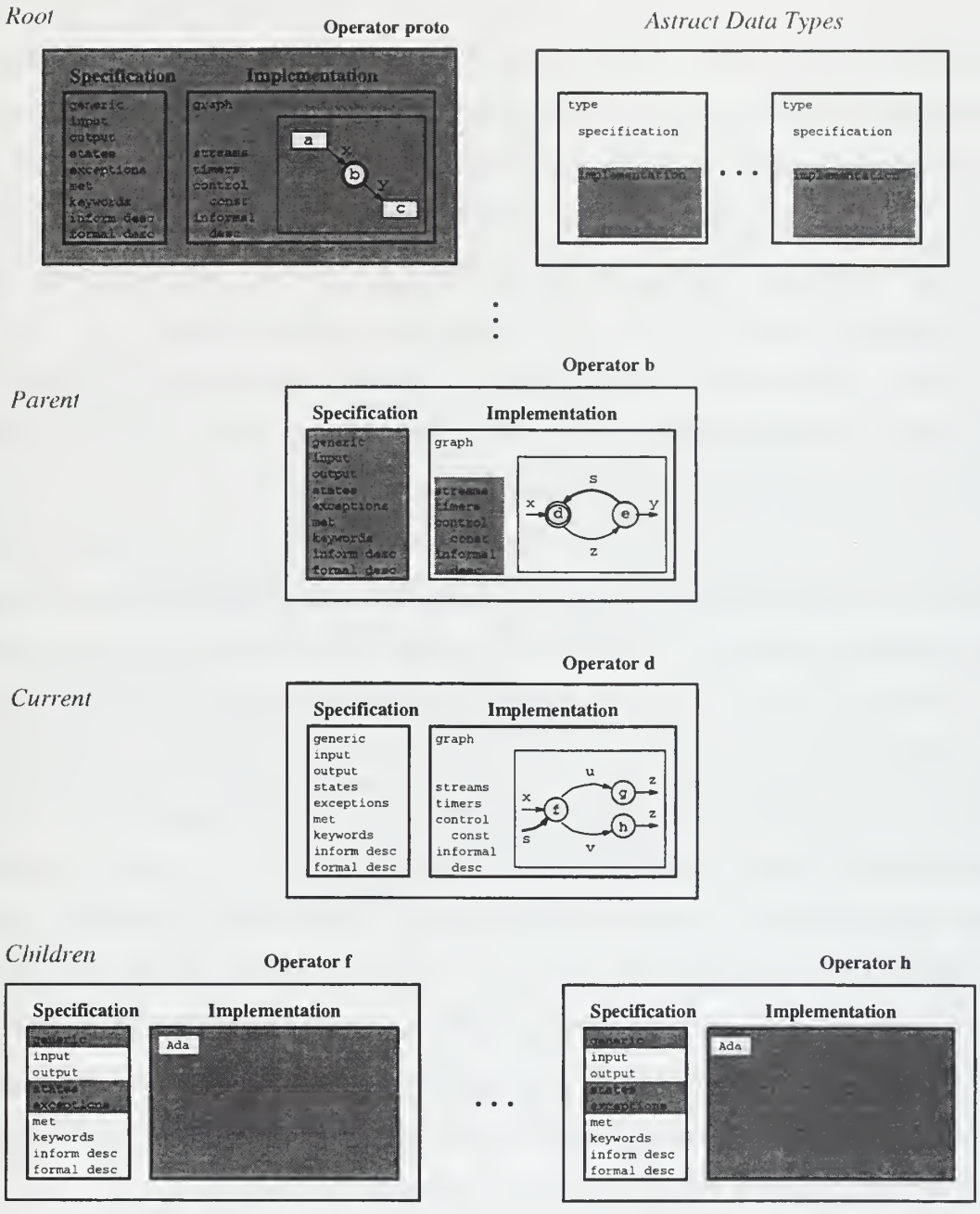


Figure 21. Relevant PSDL Tree Levels

The background checker is responsible for deriving the values to be inserted into the GRAPH_DESC data structure. Figure 22 provides the GRAPH_DESC data structure extracted from ge_interface.h. Within the GRAPH_DESC data structure, the background checker provides several references to the prototype levels just discussed, starting with the current operator being identified by cur_op_name and cur_op_num. From this starting point, all children are readily available through the data flow diagram. However, the parent can not be derived from the current operator. The background checker provides a reference to both the parent, parent_op_name and parent_op_num, as well as the root, root_op_name and root_op_num.

The current operator's specification is supported using several symbols. The cur_op_spec provides the PSDL code. The specification construct must be derived by the background checker. The input and output constructs are derived from the parent operator's data flow diagram. This was seen in Figure 19, where the input and output streams of operator d could be derived from the data flow diagram of the parent operator, operator b. In this case, streams x and s were the input streams and z was the output stream. In order to minimize the PSDL syntactical knowledge required by the graph editor, the background checker provides a partial parsing of the operator's specification. The operator's input stream list, input_list, output stream list, output_list, and maximum execution time, cur_op_spec_met and cur_op_spec_met_unit, are all provided as objects which can be operated on by the graph editor. Note that a portion of the current operator's specification is dependent upon the current operator's implementation. For example, the list of states provided in the specification are derived from the current data flow diagram. For such cases, the PSDL code provided in cur_op_spec is not maintained while in the graph editor. The background checker will update these constructs as requested.

The current operator's implementation is composed of several objects. The majority of the operator's implementation is provided in the data flow diagram. In addition, a list of timers, timer_list, and the data flow diagram's informal descrip-

```
/************************************************************ */
/* typedef for the graph description structure               */
/************************************************************ */
typedef struct graph_desc_node{
  /* From SDE to GE */
  char* root_op_name;          /* name of the root operator                  */
  int   root_op_num;           /* unique op_num of the root operator         */
  char* parent_op_name;        /* name of the parent of the current operator */
  int   parent_op_num;         /* unique op_num of the parent operator       */
  char* current_op_name;       /* name of the current operator
                                  whose dataflow graph is being edited       */
  int   current_op_num;        /* unique op_num of the current operator      */

  /* From SDE to GE */
  ST_LIST input_list;          /* list of input streams                      */
  ST_LIST output_list;         /* list of output streams                     */
                               /* NOTE: only label, label_font, stream_type_name,
                                  state_initial_value, is_state_variable are
                                  meaningful in the input_list and output_list */

  /* From SDE to GE */
  int     cur_op_spec_met;         /* MET is kept separate from the spec     */
  int     cur_op_spec_met_unit;    /* interface.  Still, only the reqmts can */

  /* MTS 11/25/96
     change cur_op_is_terminator from int to BOOLEAN */
  BOOLEAN     cur_op_is_terminator;

  /* Bi-directional between SDE and GE */
  char*   cur_op_spec;             /* Specification of current operator which */
                                   /* is edited with mini-sde.                */

  /* Bi-directional between SDE and GE */
  OP_LIST operator_list;
  ST_LIST stream_list;
  ID_LIST timer_list;
  char*   graph_informal_desc;

  /* From SDE to GE */
  ID_LIST avail_impl_langs;        /* An ID_LIST of available languages from */
                                   /* which an operator can be implemented   */

  /* Bi-directional between SDE and GE */
  char*   global_types;            /* SDE output of all types                */
} GRAPH_DESC_NODE, *GRAPH_DESC;
```

Figure 22. GRAPH_DESC Extract

tion, graph_informal_desc, are supported. The operator's data flow diagram is specified by a linked list of operators, operator_list, and a linked list of streams, stream_list.

All abstract data types are provided as PSDL code in global_types. This implementation of the graph editor does not provide graphical user interface support of abstract data types. However, the PSDL code, as derived by the background checker is made available to the graph editor.

Upon returning control back to the background checker, the ge_interface.h data structures are used to update the PSDL prototype components. Primarily, this involves updating the current and child operators. However, global updates of components may be required to maintain consistency with changes introduced in the current operator.

D. PSDL VALIDATION AND GENERATION

The PSDL Editor ensures a syntactically correct prototype. In the CAPS Release 1 PSDL Editor, all of the syntactical knowledge was embedded in the syntax-directed editor. This was appropriate since, other than the data flow diagram, the prototype was edited through the syntax-directed editor. With the focus of this implementation being on the graph editor and the graphical user interface, the prototype is now fully defined from the graph editor. Even with this change of focus, it is still possible to maintain the majority of the syntactical knowledge within the background checker. However, with the editing of the prototype being accomplished in the graph editor and the syntactical verification being performed in the background checker, a time delay is inserted between the entry of the prototype data and the detection of any errors. The length of this time delay is dependent upon how the graph editor and background checker interface. In the CAPS Release 1 PSDL Editor implementation, the syntax-directed editor generated by the Synthesizer Generator performed incremental validation of PSDL syntax as the user typed [Ref. 8]. For this

implementation, the interface between the background checker and the graph editor is accomplished similar to batch processing. In this mode of operation, an operator is edited without assistance from the background checker. Upon the user request to verify the prototype, or upon changing the current operator to another operator, the background checker is called to perform syntax/semantic validation.

There are two categories of errors which are detected by the PSDL Editor: syntactic and semantic. With automatic PSDL code generation by the background checker, syntactical errors are limited to user supplied input. Semantic errors are generally more global, typically involving one or more operators. Typically, semantic errors which are detected are based on a relationship between two symbols. With this distinction between the nature of the two categories of errors, it seemed reasonable to move the detection of syntactical errors to the graph editor, avoiding any delays introduced with a batch mode of operation, while semantic error detection being performed within the background checker. With the current architecture of the background checker and the graph editor, certain semantic errors can only be detected from the background checker, since only the background checker has a global perspective of the prototype.

In addition to the generation of the PSDL constructs associated with user supplied inputs, the PSDL Editor supplies automatic code generation for redundant data. This feature is most apparent in the generation of PSDL code to support both the specification and implementation constructs of an operator.

1. Validation of PSDL Constructs by the Graph Editor

The graphical user interface of the PSDL Editor facilitates the validation of user supplied PSDL constructs. Validation of user inputs are performed prior to updating the graph editor internal data structures. In this manner, all user supplied inputs are validated prior to being relayed back to the background checker. All fundamental PSDL data objects listed in Table IX are validated by the graph editor. Table XII provides the routines and state machines, as applicable, in which the funda-

Table XII. Fundamental PSDL Data Object Validation

PSDL Data Object	Rule	State Machine	Routine
`id`	41	N/A	`valid_id`
`id list`	11	N/A	`valid_id`, checked individually
`operator id`	24	Figure 23	`valid_op_id`
`type name`	10	Figure 24	`valid_type_name`
`integer literal`	43	N/A	`valid_integer_literal`
`text`	49	N/A	Motif
enumeration			Motif

mental data objects are validated. Validation routines are found in `ge_utilities.c`, which is included in Appendix D, starting on page 202. All enumeration values listed in Table X are validated by the graphical user interface by only allowing the user to select one value from a predefined list.

For the complex PSDL data objects listed in Table XI, the graph editor does not contain the syntactical knowledge to validate user supplied input. While required,

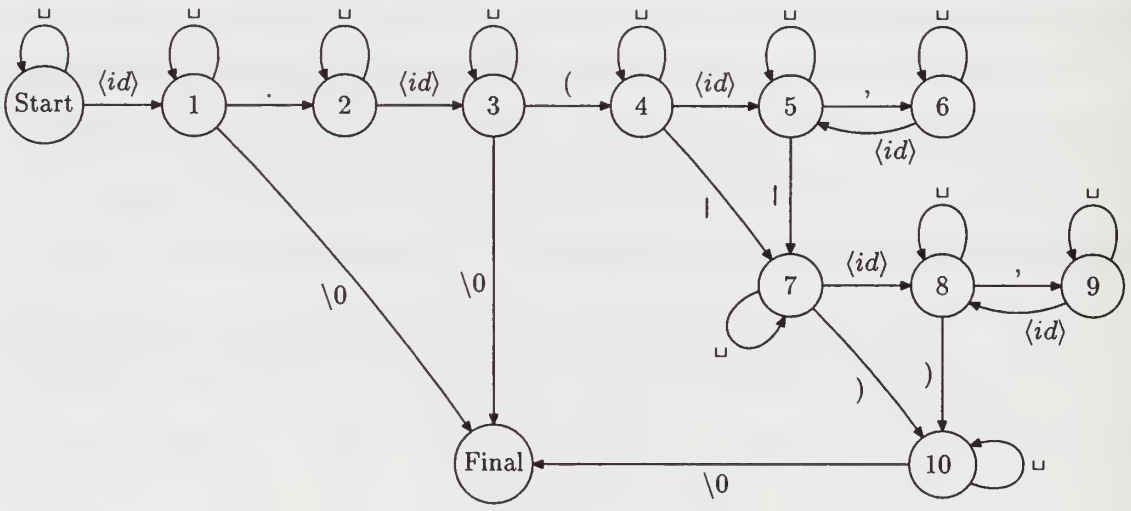


Figure 23. ⟨*op_id*⟩ Validation State Machine

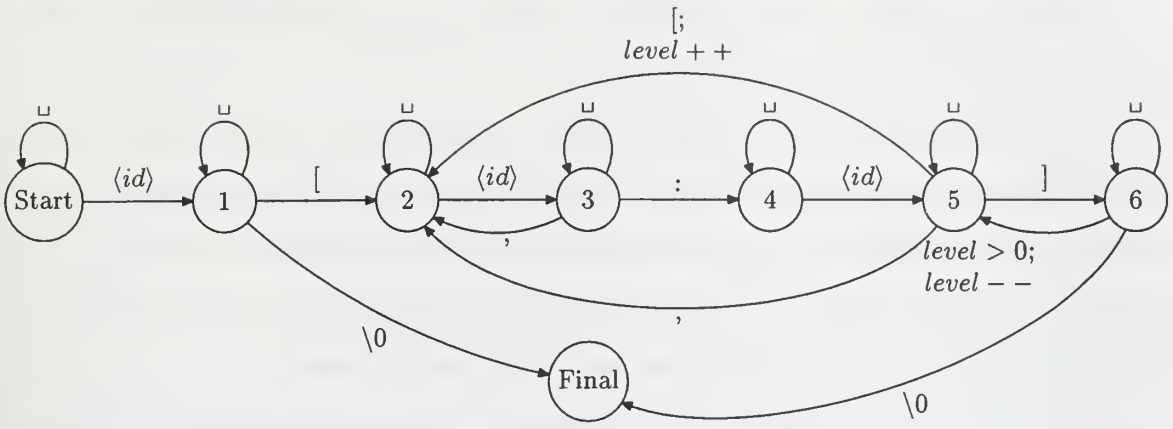


Figure 24. ⟨*type_name*⟩ Validation State Machine

the facilities needed to provide this validation have yet to be implemented.

2. PSDL Redundant/Derived Data

The operator's specification defines the external interface of the operator while hiding the details of the implementation. The specification provides a "summary" of the operator, and thus contains redundant information found in the operator's implementation. The PSDL Editor provides support for this redundant information. The user is only required to enter data once. The background checker is equipped with the semantic knowledge required to derive the redundant data. In addition to saving the user from entering redundant information, this implementation avoids semantic errors due to inconsistencies between redundant data. If data is entered twice, there is the possibility that it will be inconsistent. If it is only entered once, there is no possibility for inconsistent data.

The graphical user interface designed for the graph editor has been limited to the single entry of redundant data. There is no semantic knowledge regarding redundant data in the graph editor. All redundant data processing is performed by the background checker.

3. PSDL Semantic Consistency by the Graph Editor

While limited by the local scope of a single operator, the graph editor is still capable of enforcing some semantic consistency.

Within the scope of an operator, those operators used to implement the data flow diagram must be unique. Uniqueness of operator labels does not apply to operators of an abstract data type. Uniqueness is enforced upon user entry of the operator label. The routine `unique_op_id`, found in `graph_object_list.C` (reference Appendix D, starting on page 277), is used to test for uniqueness.

The characteristic of an operator being implemented as a composite terminator (i.e., having a maximum execution time of zero) is inherited by all operators of the data flow diagram. The graph editor enforces this inheritance by limiting the user to the use of only terminators within the graphical user interface.

Unlike the scoping rules for operator labels, which must be unique for non-type operators within a data flow diagram, multiple uses of a stream are allowed within a data flow diagram. While each operator implements its own type of stream (i.e., data flow stream or sampled stream), streams also contain global characteristics. These include stream type, state stream, and initial value. The graph editor provides for the propagation of these characteristics for all streams within the data flow diagram. It is the requirement of the background checker to maintain consistency of these global characteristics external of the current operator. The graph editor enforces consistency through the routine `propagate_stream`, found in `graph_object_list.C` (reference Appendix D, starting on page 277).

With a global perspective, the background checker is capable of providing additional semantic validation. Specifically, the background checker has provisions for validating the input and output streams specifications in a child/parent relationship. As was seen previously in Figure 19, the data flow diagram of the parent operator defines the inputs and outputs to the child operator. When a child operator is implemented with a PSDL network, the external input and output streams present in the

child's data flow diagram must exactly match the input and output streams found in the parent's data flow diagram [Ref. 3]. External input and output streams which were not defined in the parent's data flow diagram are flagged as being illegal by the background checker. Input and output streams defined in the parent's data flow diagram but missing from the child's implementation are flagged as being missing.

In these two cases, it is beyond the capability of the PSDL Editor to determine a corrective action. Instead, upon detecting the error, the background checker provides an error message to the user. Facilities are provided to navigate directly to the parent or child operator in order to correct the problem.

The background checker provides an additional test to warn against a semantic violation. As was previously mentioned in Chapter II, PSDL has no global variables. However, as currently implemented in CAPS Release 1, streams are global, based on the stream name. The background checker tests for the use of global streams and provides a warning on its use.

Figures 25 through 27 provide an example of these error conditions which are detected by the background checker. Figure 25 provides the parent data flow diagram. Figure 26 provides the decomposition of operator `b` found in the parent's data flow diagram. Figure 27 depicts the resulting error messages from this prototype. In this case, the parent's data flow diagram specifies that operator `op_b` shall have an input, `op_b_in`, and an output, `op_b_out`. Within the implementation of operator `op_b`, found in Figure 26, two undefined, external streams are specified, called `in` and `out`. These two errors are reported as the first two errors in Figure 27. Next, the lack of input `op_b_in` and output `op_b_out` from the child's implementation are reported as the next two errors in Figure 27. Finally, the global stream `x` is defined both in the parent's data flow diagram and the child's data flow diagram, without the stream being passed between the parent and the child. The final two error messages of Figure 27 report this condition.

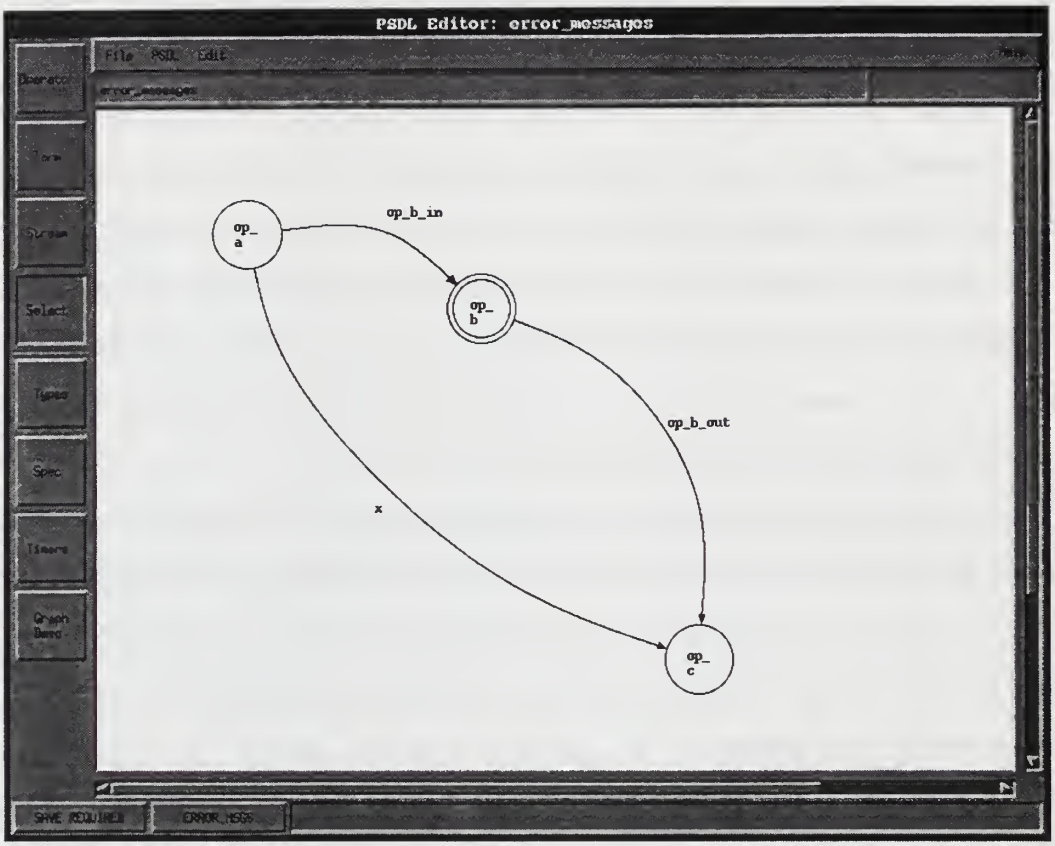


Figure 25. Parent Graph Depicting Errors

E. PSDL SYNTAX CHANGES

Several modifications have been made to the PSDL grammar in support of CAPS Release 2. The new PSDL Editor is required to support these changes. A copy of the CAPS Release 2 PSDL grammar is provided as Appendix A. A few small fixes have been incorporated, such as updates to the ⟨*letter*⟩ and ⟨*alpha_numeric*⟩ production rules. Previously, these rules allowed any number of consecutive underscore characters ('_') in an ⟨*id*⟩. The grammar rules have been changed (reference production rules 47 and 48 in Appendix A) to be consistent with the construct of identifiers in Ada. Another small change was the generalization of the implementation language identifier in the ⟨*type_impl*⟩ and ⟨*operator_impl*⟩ production rules (reference rules 17

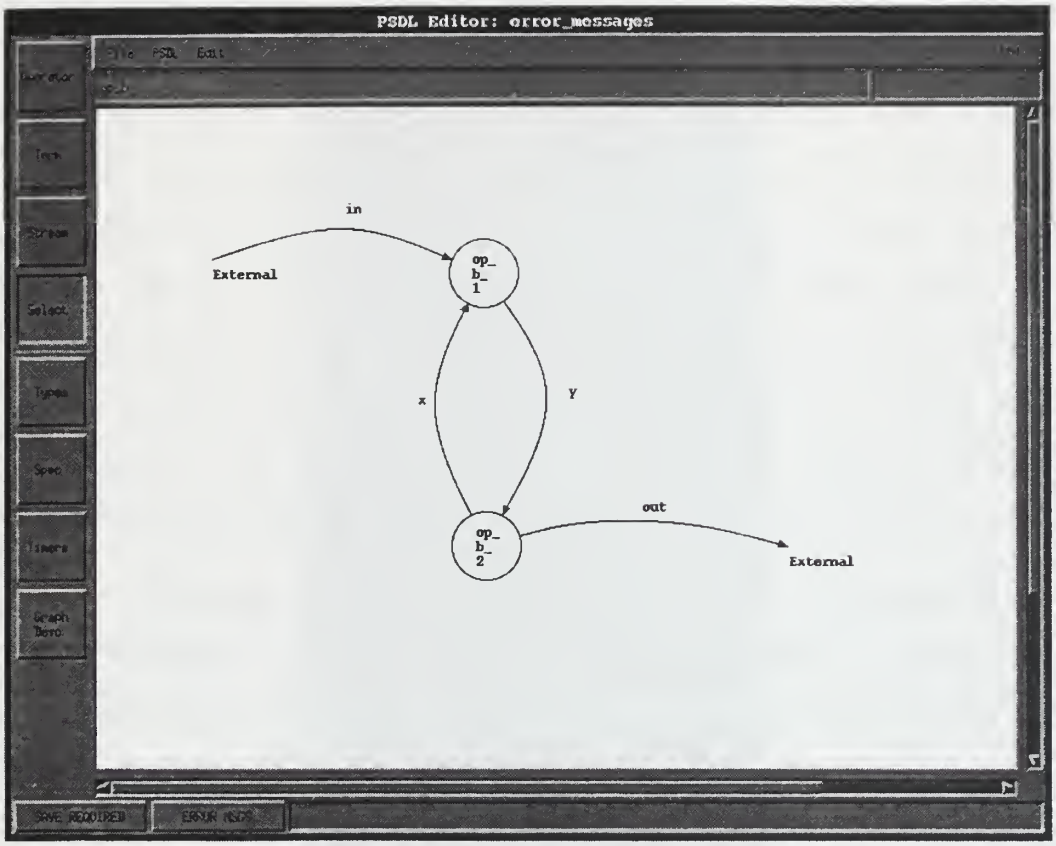


Figure 26. Child Graph Depicting Errors

and 18 in Appendix A). Previously, only Ada was allowed as an implementation[22].

Two more substantial changes were made to the PSDL grammar which impact the PSDL Editor. From the standpoint of the user interface, these changes are transparent to the user. However, both require support from the PSDL Editor to implement.

1. PSDL Data Flow Diagram Properties

In CAPS Release 1, the PSDL file generated by the PSDL Editor was sufficient to support the Execution Support tools, yet was not sufficient to support the PSDL

[22]Currently, the only supported implementation languages are Ada and TAE. Facilities have been provided in the grammar for future supported languages.

Figure 27. Syntax-Directed Editor Error Messages

Editor. The PSDL Editor provided an additional file, with an extension of ".grf", to facilitate the presentation of the data flow diagram. In CAPS Release 1, the PSDL grammar did not have provisions to capture the full specification of the data flow diagram (i.e., the visual presentation requirements). In CAPS Release 2, the PSDL grammar has been updated with a property construct (reference production rules 21, 22, and 23 in Appendix A) for this purpose. This change provides for the tagging of properties to vertices and edges of a data flow diagram based on predefined identifiers. These properties capture the information which was previously recorded in the ".grf" file. A list of these identifiers was provided in Chapter II as Table VI (page 38). Currently, these properties are only used by the PSDL Editor.

2. Unique Identifier Suffixes in CAPS Release 2

CAPS Release 2 introduced a new semantic convention for the purpose of enforcing PSDL scope rules. Professor Berzins described the conventions used in his email dated 25 July 1996, which has been provided as Appendix C.

The semantics of PSDL requires that the operators within a composite PSDL operator (i.e., an operator which has been implemented with a PSDL network) be local to that operator. The scoping of operators is illustrated in Figure 28, a portion of a PSDL prototype, in which operators `fm_a` and `fm_b` both contain operators with common names, `x`, yet are local to their respective parent operator. However, this only applies to PSDL operators. Scoping for operators of a PSDL type are global, as PSDL types are globally available within the prototype heirarchy. Thus, `t.x` refers to the same operator of `t`, from both `fm_a` and `fm_b`. Note that within an operator, PSDL operator names must be unique, as was described previously.

In order to implement the PSDL scoping rules, unique integer suffixes are attached to the PSDL operator identifiers, thus ensuring that all PSDL operators are unique. In addition to supporting the scoping rules of PSDL operators, suffixes are also used to support multiple instances of PSDL type operators within the same graph. Thus each operator, both PSDL operators as well as PSDL type operators,

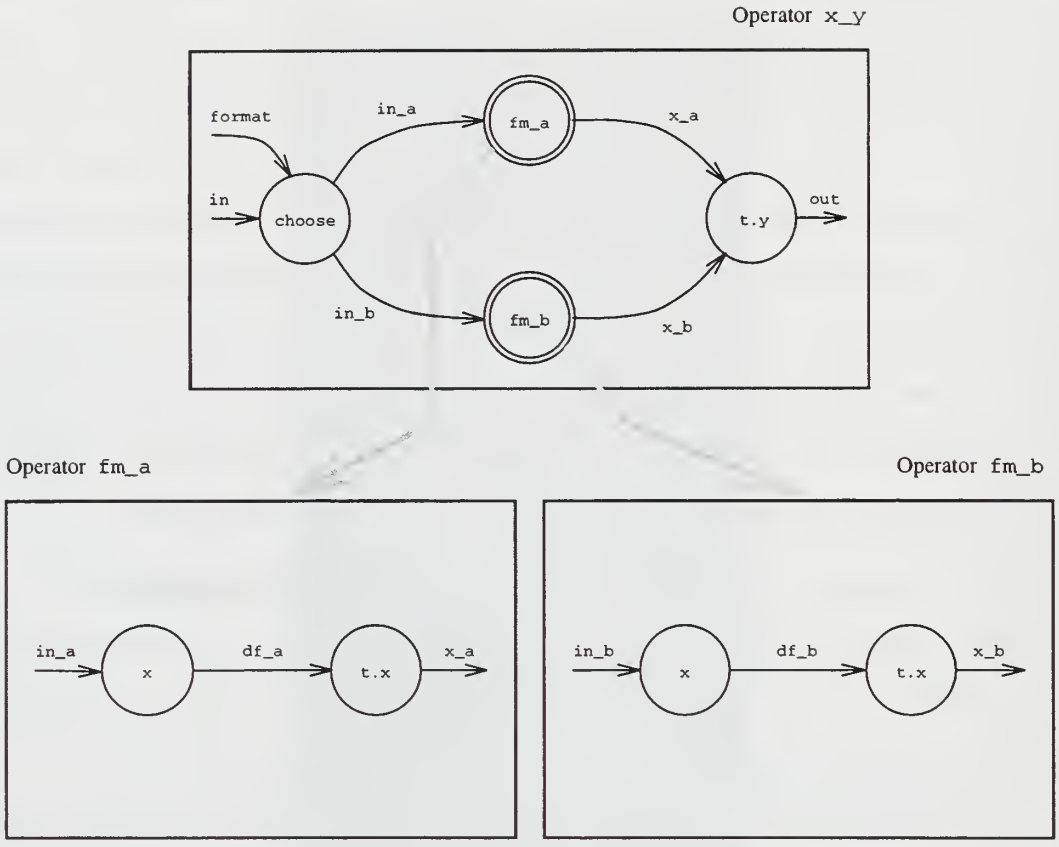


Figure 28. Scope of PSDL Operators

are assigned a vertex number within a data flow diagram. In addition, each PSDL operator is assigned an operator number. The convention is as follows:

⟨*op_name*⟩_⟨*op_num*⟩_⟨*vertex_num*⟩
⟨*type_op_name*⟩_⟨*vertex_num*⟩

The above prototype portion is depicted again in Figure 29, in which suffices have been included.

Note that this is actually not a syntax change. The inclusion of operator and vertex number suffixes is within the syntax of an ⟨*id*⟩ (reference production rule 41 in Appendix A). The use of suffixes to implement the PSDL scoping rules is a semantic convention which is used by the CAPS subsystems. The implementation of the suffix

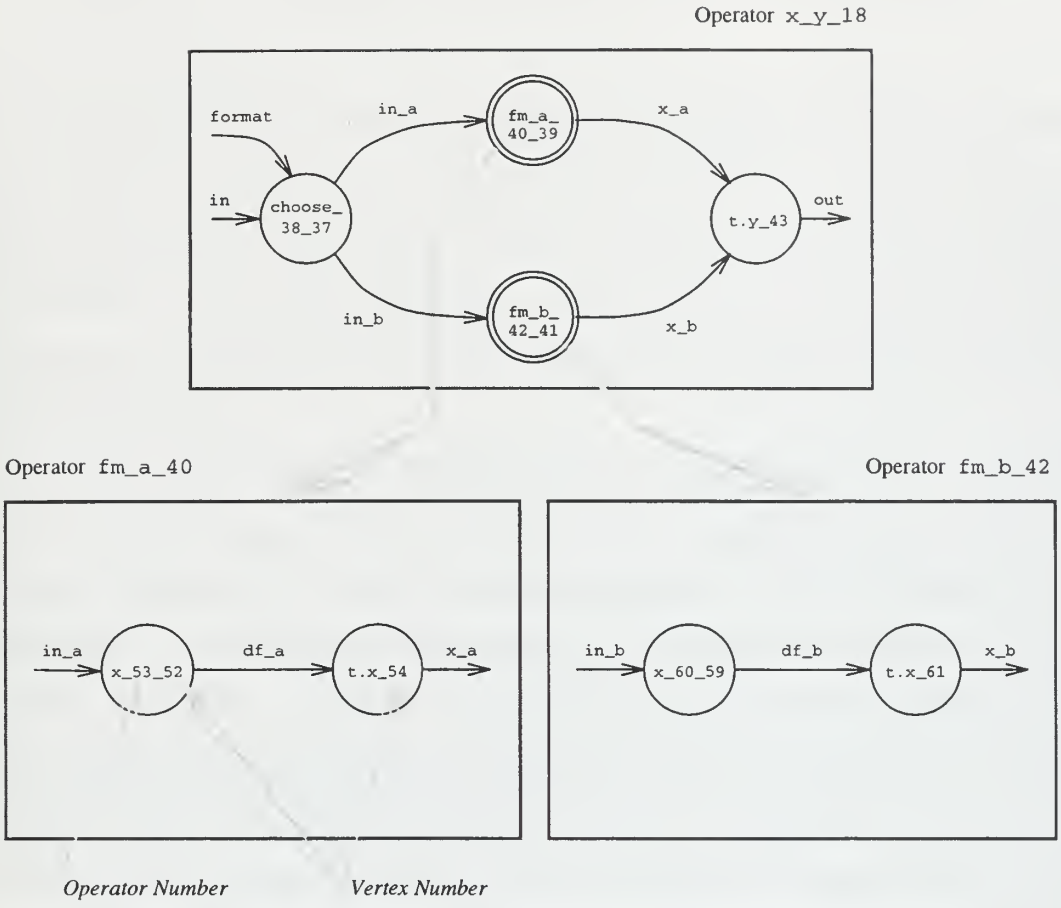


Figure 29. PSDL Operator Suffixes

convention is to be hidden from the PSDL Editor user, thus requiring the PSDL Editor to fully automate its suffix behaviour.

IV. USER-INTERFACE DESIGN

The core of a PSDL prototype is the data flow diagram. In the CAPS Release 1 PSDL Editor, the graph editor was solely used to edit/view the data flow diagram. Following the editing of the data flow diagram, the syntax-directed editor was used to edit all other constructs of the prototype. This release of the PSDL Editor attempts to streamline the editing process of a PSDL prototype by focusing the user interaction to the graph editor.

From an inspection of the PSDL grammar (reference Appendix A), it can be seen that a large number of PSDL constructs can be synthesized by the editor, using simple user provided data objects (e.g., literal values, text strings, identifiers, and list of identifiers). As a PSDL prototype consists of a network of operators connected by streams, the PSDL constructs, and hence the user provided data objects, are all associated with either an operator or a stream. The only exception being abstract data types. Thus, a graphical user interface model was developed, which build upon the earlier data flow diagram effort of the CAPS Release 1 graph editor, with the addition of pop-up windows associated with operators and streams for the entry of user provided data objects.

Complex PSDL constructs (e.g., expressions), as discussed in Chapter III, are associated with operators and streams as well. However, the complexity of these constructs does not lend themselves to a simple synthesis from user provided data objects. As an initial attempt of expanding the graphical interface of the PSDL Editor, the specification of these constructs was maintained in PSDL syntax.

A large portion of the graphical interface produced by Captain Robert Dixon, USMC [Ref. 13], was maintained in this implementation. Primarily, the interface used for the data flow diagram was preserved, with a few enhancements. Expansion of the graphical interface was partially outlined as part of the NPS CS4520 (AY96Q4) class project, implemented by Mr. Douglas Lange and Mr. Dagohoy Anunciado.

Additional modifications and enhancements to the interface were a result of this research.

A. PSDL EDITOR ENVIRONMENT

The graphical user interface provided in CAPS Release 1 (i.e., the graph editor's interface) was built upon the X Window System[23], using the Motif[24] widget set. The selection of Motif for use in the CAPS Release 1 PSDL Editor was discussed in Captain Dixon's thesis [Ref. 13]. Motif provides assistance in the development of an application by providing a standard look and behavior to the user interface.

1. PSDL Editor Layout

The PSDL Editor's graphical user interface is designed to facilitate the specification of a PSDL operator. As was discussed previously, the user interface is limited to a single PSDL operator at any given time. The PSDL Editor does not provide for the implementation language (e.g., Ada) editing of an operator.

The PSDL Editor's graphical user interface is depicted in Figure 30. The graphical user interface consists of six areas, which are identified in Figure 30 and discussed in the following paragraphs. Pop-up windows are used to support the specification of constructs directed at streams and operators of a data flow diagram.

a. Main Window

This is the X Window from which the user interacts with the PSDL Editor. The MAIN WINDOW is equipped with a title along with several control widgets on the top bar. The title is composed of "`PSDL Editor:`" and the name of the prototype. The prototype name is based on the file name (i.e., name of root operator) provided upon execution of the PSDL Editor. If no file name is provided, the title is set to `DEFAULT_PROTO_NAME`. The operation of the control widgets is based on the X

[23]X Window System is a trademark of the X Consortium.

[24]Motif is a trademark of the Open Software Foundation.

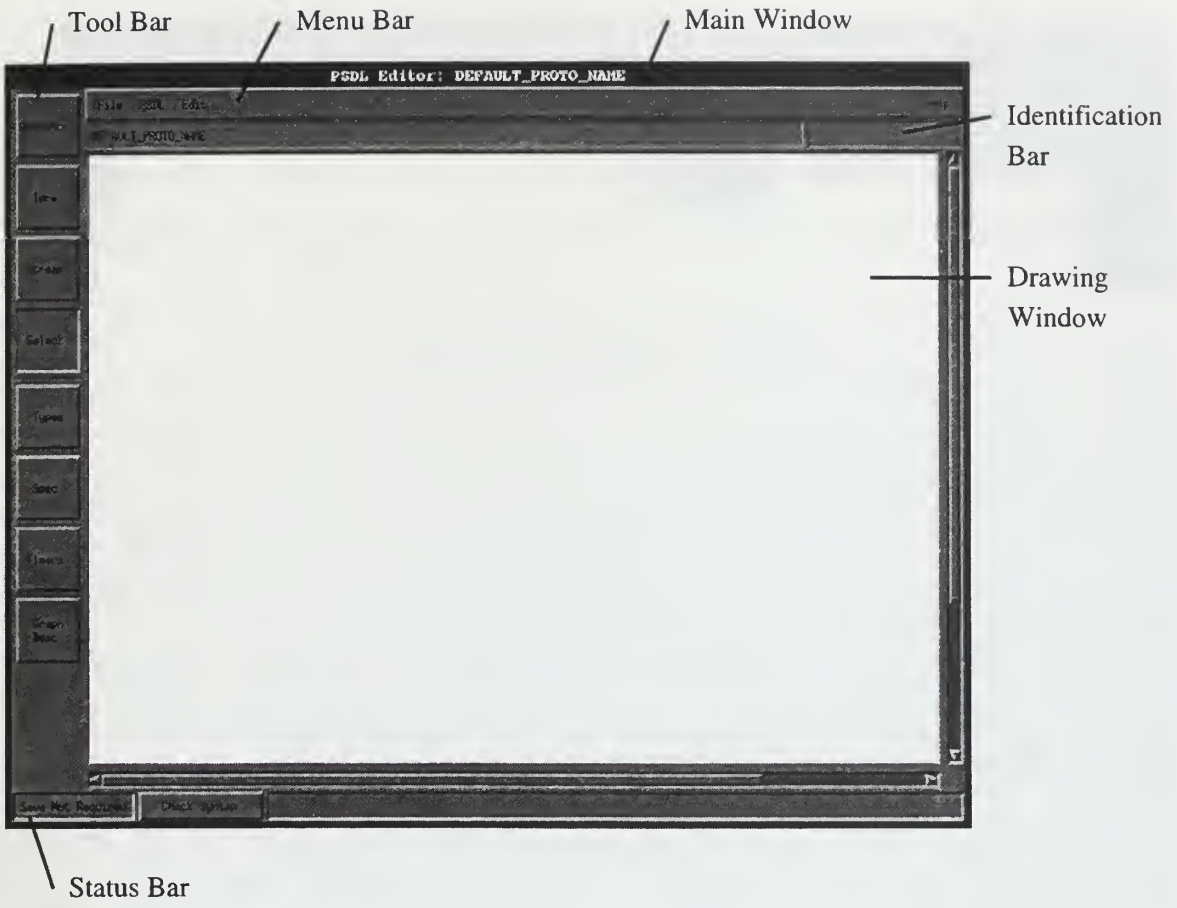


Figure 30. PSDL Editor Layout

Window System initialization. Reference Appendix E for notes on customization of the MAIN WINDOW.

b. Menu Bar

Pull-down menus provide easy access to PSDL Editor functions. Standard access to the pull-down menus is supported, accomplished using the left-mouse button. Figure 31 depicts all four pull-down menus. Note that the MENU BAR was distorted in the generation of this figure in order to display all four menus simultaneously. Only one menu is available at a time in the PSDL Editor.

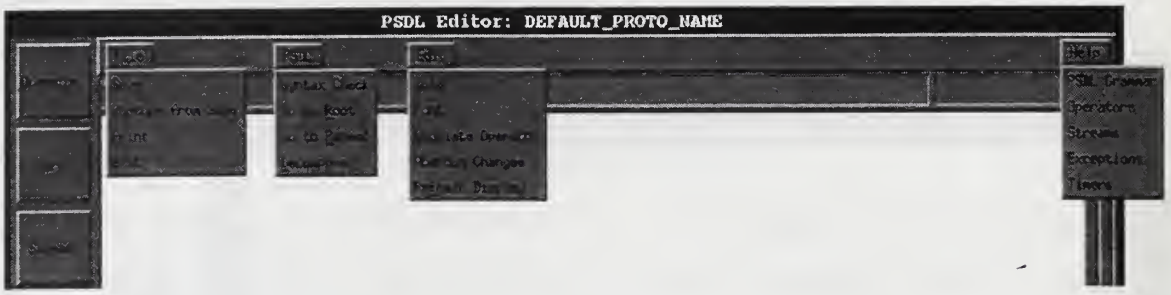


Figure 31. PSDL Editor Menus

c. Tool Bar

The TOOL BAR is divided into two areas by a horizontal line. Above the line, buttons are provided to access tools for "drawing" the data flow diagram. Below the line, buttons are provided to access functions for editing several PSDL constructs. The functions provided here are relative to the current operator or are global in nature (e.g., the function TYPES is directed towards abstract data types). As will be seen later, other functions, available through pop-up windows, are relative to the operators and streams contained in the data flow diagram. The functions here (i.e., SPEC, TIMERS, and GRAPH DESC) are directed to the operator indicated in the IDENTIFICATION BAR.

The selected drawing tool, above the horizontal line, is independent of any tool selected below the horizontal line. While drawing functions are not active during the use of a tool below the horizontal line, upon exiting, the previously selected drawing tool remains in effect until another drawing tool is selected.

d. Identification Bar

The IDENTIFICATION BAR provides the name of the current operator (left data window) as well as the maximum execution time of the current operator (right data window). These values can not be modified at this level. Instead, the user is required to change the values from the parent operator's data flow diagram. The values displayed from the root operator can not be changed.

e. Drawing Window

This area is used to display/edit the data flow diagram. Scroll bars are provided as necessary to view the entire data flow diagram window. Beyond the area accessed through the window scroll controls, the DRAWING WINDOW can not be enlarged. If a larger drawing area is required, consider decomposing operators.

f. Status Bar

The STATUS BAR provides feedback to the user. Divided into three windows, the STATUS BAR provides: (1) an indication that the prototype has been modified (SAVE REQUIRED versus SAVE NOT REQUIRED), (2) an indication that the prototype's syntax should be verified (CHECK SYNTAX versus ERROR MSGS), and (3) a window for displaying status and error messages.

The first two indicators are button widgets. The labels on the buttons are changed as required to provide feedback. Activation of the button provides a short cut to the desired function (i.e., save the prototype, perform a syntax check, and display any error messages). The last indicator provides an area to provide editor feedback. Typically, an information pop-up window is used to display an editor message. This window is used to remind the user after the pop-up window has been acknowledged. The window is automatically erased, typically on the next operation.

2. Component Identification

From the MAIN WINDOW, the user can access an Operator Property or Stream Property pop-up window. This is accomplished by positioning the cursor over an object in the DRAWING WINDOW and pressing the right-mouse button. The appropriate pop-up window will be accessed. Figures 32 through 34 provide an example of the these three windows. Component[25] Identification numbers are provided, which reference into Table XIII (Index), providing identification of all displays/controls.

[25]In this case, component refers to a Motif widget and not to a PSDL component.

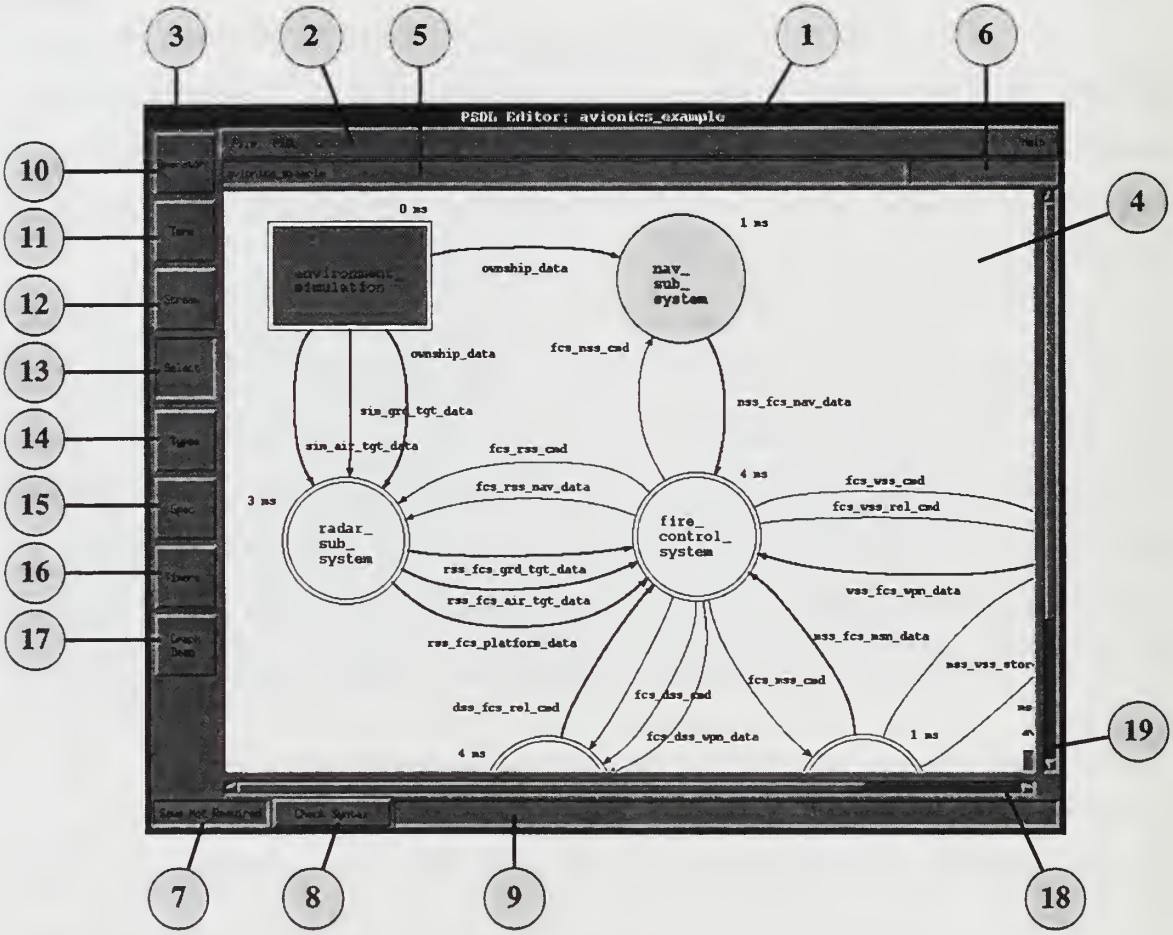


Figure 32. PSDL Editor Component Identification

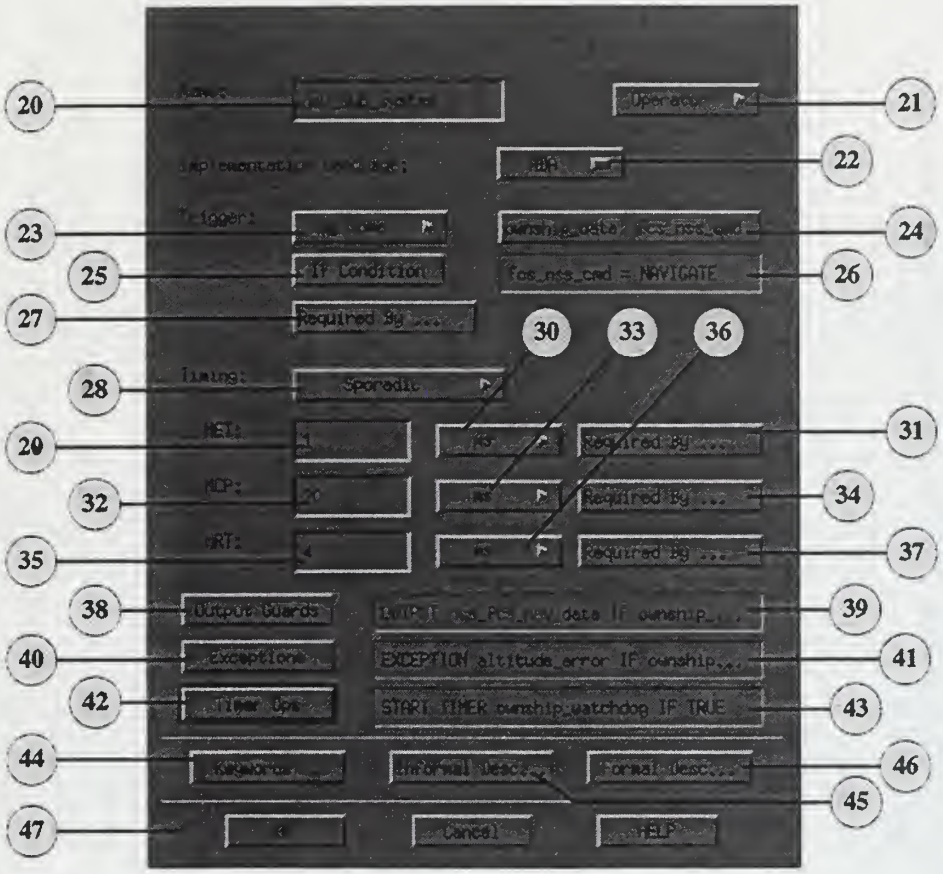


Figure 33. PSDL Editor Operator Pop-up Component Identification

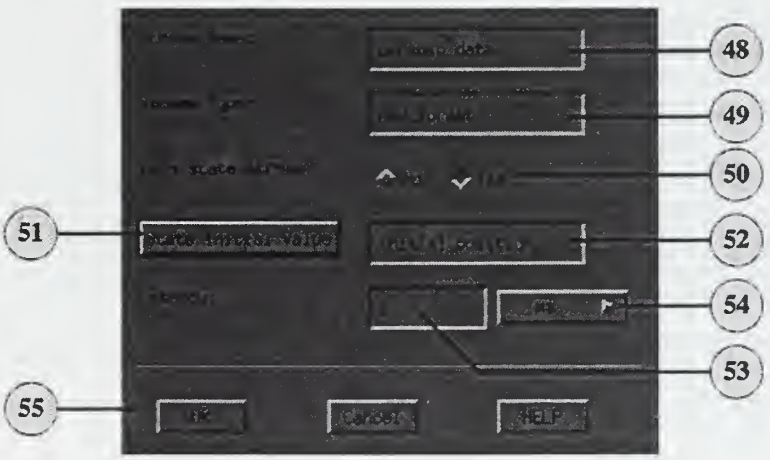


Figure 34. PSDL Editor Stream Pop-up Component Identification

Table XIII.: PSDL Editor Component Identification

Window	Index	Identification	Component Type	Indicator	Hides	Validation
Main	1	Prototype Name	Display Only	N/A		N/A
	2	Menu Bar	Control	N/A		N/A
	3	Tool Bar	Control	N/A		N/A
	4	Drawing Window	Data Flow Diagram	Drawing		`valid_op_id`, `unique_op_id`, `valid_id`
	5	Current Operator Name	Display Only	Data		N/A
	6	Current Operator MET	Display Only	Data		N/A
	7	Save Button/Indicator	Control	Label		N/A
	8	Check Syntax Button/Indicator	Control	Label		N/A
	9	Status Message Window	Display Only	Data		N/A
	10	DFD Operator Tool	Control	N/A		N/A
	11	DFD Terminator Tool	Control	N/A		N/A
	12	DFD Stream Tool	Control	N/A		N/A
	13	DFD Select Tool	Control	N/A		N/A
	14	Types Tool	Control-Text	N/A		Not Validated
	15	Specification Tool	Control-Text	N/A		Not Validated
	16	Timers Tool	Control-IdList	N/A		`valid_id`
	17	Graph Informal Description Tool	Control-Text	N/A		Motif
	18	DFD Horizontal Scroll Control	Control	N/A		N/A
	19	DFD Vertical Scroll Control	Control	N/A		N/A

Table XIII.: PSDL Editor Component Identification

Window	Index	Identification	Component Type	Indicator	Hides	Validation
Operator	20	Operator Name	Data Entry	Data		`valid_op_id, unique_op_id`
	21	Operator/Terminator Selection	Select	Enumeration		Motif
	22	Implementation Language	Select	Enumeration		Motif
	23	Trigger	Select	Enumeration		Motif
	24	Trigger Identifier List	Control-IdList	Data...	Yes	`valid_id`
	25	Trigger If Condition	Control-Text	N/A		Not Validated
	26	Trigger Id Condition Expression	Display Only	Data...	Yes	N/A
	27	Trigger Required By	Control-IdList	Label...		`valid_id`
	28	Timing	Select	Enumeration		Motif
	29	MET Value	Data Entry	Data	Yes	`valid_integer_literal`
	30	MET Units	Select	Enumeration	Yes	Motif
	31	MET Required By	Control-IdList	Label...	Yes	`valid_id`
	32	MCP/Period Value	Data Entry	Data	Yes	`valid_integer_literal`
	33	MCP/Period Units	Select	Enumeration	Yes	Motif
	34	MCP/Period Required By	Control-IdList	Label...	Yes	`valid_id`
	35	MRT/Finish Within Value	Data Entry	Data	Yes	`valid_integer_literal`
	36	MRT/Finish Within Units	Select	Enumeration	Yes	Motif
	37	MRT/Finish Within Required By	Control-IdList	Label...	Yes	`valid_id`
	38	Output Guard Control	Control-Text	N/A		Not Validated
	39	Output Guard Display	Display Only	Data...	Yes	N/A

Table XIII.: PSDL Editor Component Identification

Window	Index	Identification	Component Type	Indicator	Hides	Validation
	40	Exceptions Control	Control-Text	N/A		Not Validated
	41	Exceptions Display	Display Only	Data...	Yes	N/A
	42	Timer Operation Control	Control-Text	N/A		Not Validated
	43	Timer Operation Display	Display Only	Data...	Yes	N/A
	44	Keywords	Control-IdList	Label...		`valid_id`
	45	Informal Description	Control-Text	Label...		Motif
	46	Formal Description	Control-Text	Label...		Motif
	47	Exit Controls	Control	N/A		N/A
Stream	48	Stream Name	Data Entry	Data		`valid_id`
	49	Stream Type	Data Entry	Data		`valid_type_name`
	50	State Stream Selection	Select	Enumeration		Motif
	51	State Initial Value Control	Control-Text	N/A	Yes	Not Validated
	52	State Initial Value Display	Display Only	Data...	Yes	N/A
	53	Latency Value	Data Entry	Data		`valid_integer_literal`
	54	Latency Units	Select	Enumeration		Motif
	55	Exit Control	Control	N/A		N/A

3. Types of Components

Under the Component Type column of Table XIII, the various types of user interaction are identified. The PSDL Editor utilizes a limited set of widgets to interact with the user. These components are used both for editing a data object as well as viewing a data object. Typically there is not sufficient space within a window to view all of a data type. Controls are used to access additional Motif widgets from which the entire data object can be viewed/edited. In most cases, an indication is provided that additional information is available through a pop-up window.

The data flow diagram requires significant interaction. It will be discussed in another section.

a. Display Only

This widget provides no input/control functionality. It is only used to display data. Data may be obtained from the editor and displayed, as is the case for the Current Operator Name (Index 5). Alternatively, the input may be provided by the editor, but accessed from a control button, as in the case of Trigger Id Condition Expression (Index 26) which is accessed from the Trigger If Condition (Index 25).

b. Data Entry

This widget provides direct data entry through the keyboard. Prior to data entry, ensure that the mouse cursor is positioned within the Data Entry window. The user can scroll through the Data Entry window using the left and right arrow keys, as required.

c. Select

This widget provides a select-one-of function. Values act similar to an enumeration type. Values are predefined and only one value can be selected. The Select type may be either a radio widget, such as State Stream Selection (Index 50), or a pull-down version, such as Implementation Language (Index 22). The pull-down version is activated by depressing and holding the left-mouse button over the component. Move the cursor over the desired value and release the mouse button. An example of the Implementation Language widget is provided in Figure 35. Note

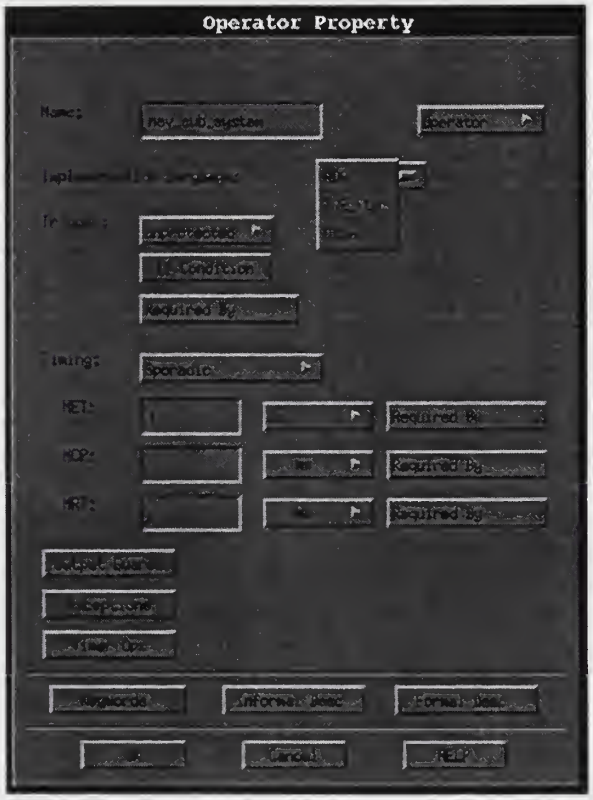


Figure 35. Select Component

that, in this example, the cursor is not visible.

d. Control

Control widgets are accessed by depressing the left-mouse button over the component.

e. Control-Text

Control-Text components access a pop-up Text window. An example of a Text Window is provided in Figure 36, in which the prototype's abstract data types were accessed through the TYPES TOOL (Index 14).

Within the Text Window, the user can position the cursor with the mouse and edit the text. The cursor must remain within the Text Window while editing the text. Scroll bars are provided for moving through the text. Initially, all changes made within the Text Window are local to the Text Window. In order

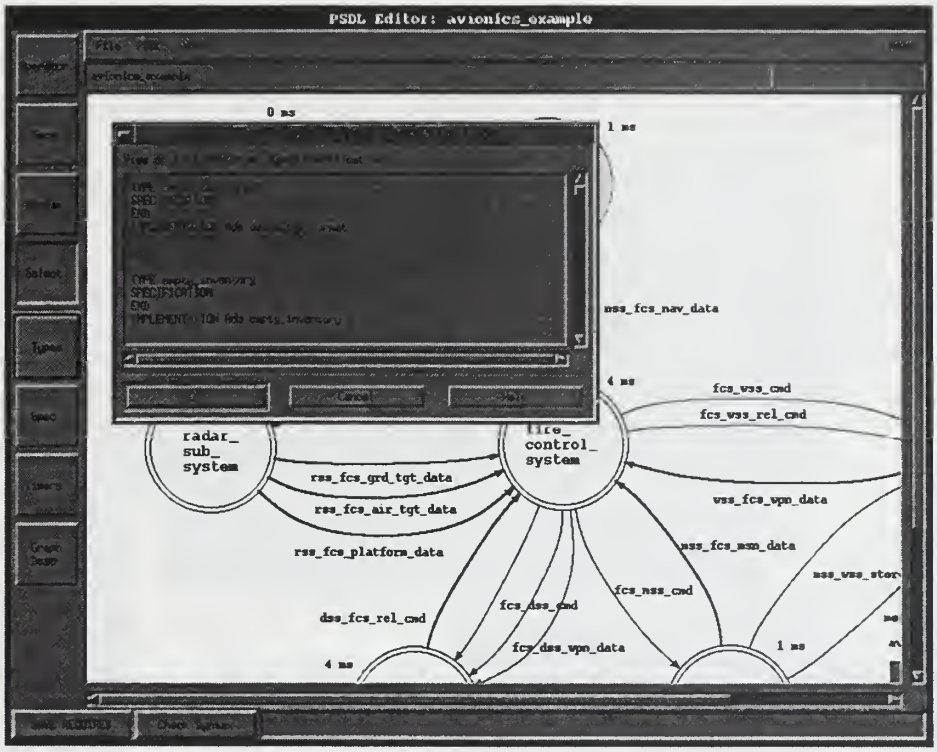


Figure 36. Text Window Component

to modify the prototype, the changes must be accepted. This is accomplished by depressing the OK button. Changes may be aborted by depressing the CANCEL button.

f. Control-IdList

Control-IdList components access a pop-up identifier list editor. Within this window, a list of identifiers may be viewed/edited. An example of an Id List Window is provided in Figure 37, in which the operator's Timer list is accessed through the TIMERS TOOL (Index 16).

From the Id List Window, the identifier list can be viewed directly. In order to add an identifier, the Add button is depressed. A pop-up window is provided to add the identifier to the list. The new identifier is accepted locally through the OK button. The addition of an identifier to the Id List is depicted in Figure 38. A similar

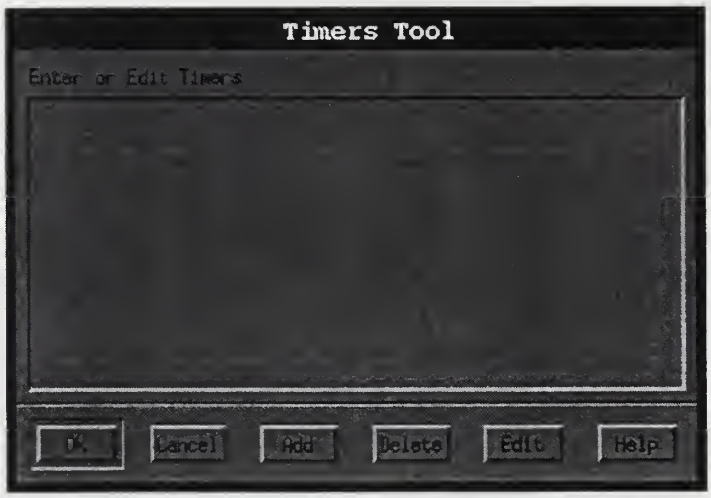


Figure 37. Identifier List Editor Component

process applies to the editing of an existing identifier. However, in the case of editing an identifier, the identifier must first be selected. Otherwise an information message will be displayed to select an identifier first. In order to delete an identifier from the list, select the identifier and depress DELETE. Again, an information message will be displayed if an identifier is not first selected.

Like the Text Window, initially, all changes made within the Id List Window are local. In order to modify the prototype, the changes must be accepted. This is accomplished by depressing the OK button. Changes may be aborted by depressing the CANCEL button.

4. Display Indications

Under the Indicator column of Table XIII, the various methods used to display data are identified. Where practical, data objects are displayed within the window. However, due to the limited space within a window, it is not always possible to display the entire value of a data object. An effort was made to indicate to the user where data is available. This is not universal, in that the tool buttons have no indication of the existence of data.

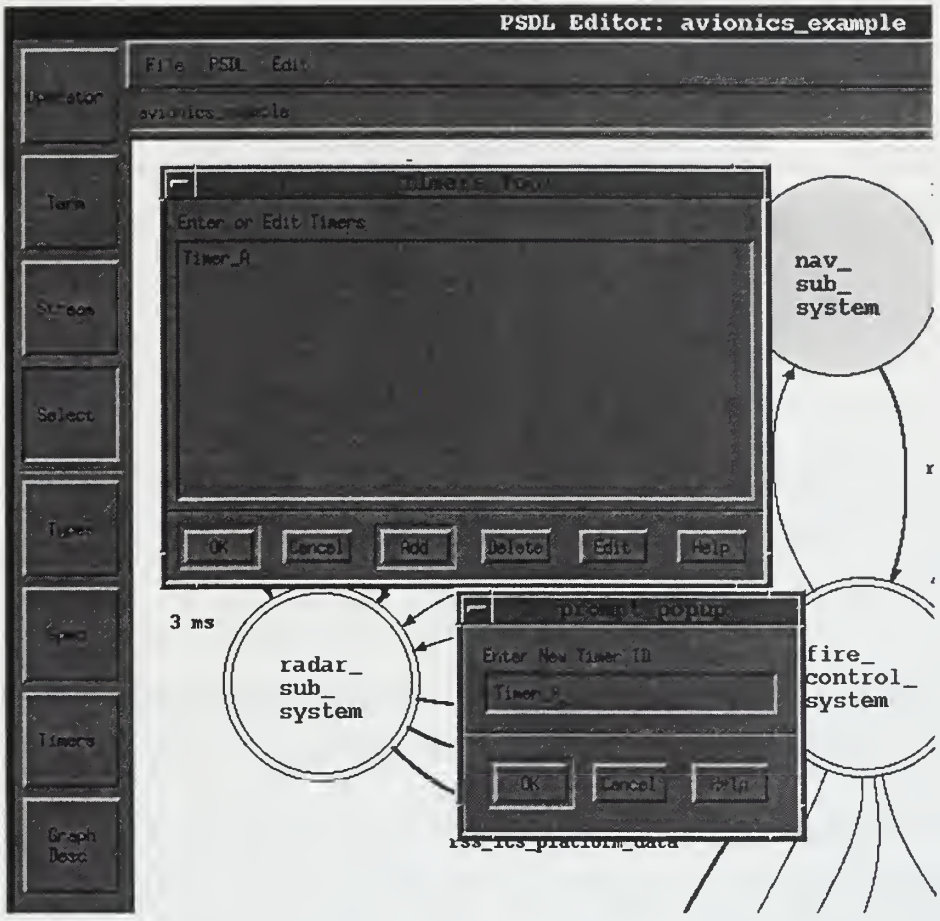


Figure 38. Adding to an Identifier List

As for the data flow diagram, it is visible in the DRAWING WINDOW. However, in order to view all attributes of the data flow diagram, the stream and operator pop-up windows must be accessed. The only method available to view the entire prototype is through the PSDL code generated by the PSDL Editor.

a. Data

A Data indication can be viewed directly from its Data or Display Only component. The user can scroll within the data window through the use of the left and right arrow keys.

b. Data...

The Data... indication is similar to the Data indicator. However, in this case, the data value is typically much larger than the display window. If the data value will not fit within the window, the data value is truncated and a set of epsilons (i.e., "...") is appended. Use the control mechanism to access the entire data value.

c. Enumeration

The current Enumeration value is indicated directly on the control component.

d. Label

A Label indication is similar to an Enumeration indication in that any feedback is provided directly on the control component.

e. Label...

In this case, much like the Data... indication, a set of epsilons (i.e., "...") is appended to the label to indicate that a data value is associated with the control component. Without the epsilons, a value has not yet been assigned.

f. Hidden

Not all data objects are available in all situations. Some data objects are context depended. For example, a state stream initial value is not relevant unless the stream is a state stream. As a method of enforcing PSDL semantics, several components are displayed based on the context of the operator/stream. The Hides column of Table XIII indicates those components which may be unavailable due to context. Figure 39 depicts the case of a stream which is not a state stream. Here, the State Initial Value Control (Index 51) is "grayed out" and the State Initial Value Display (Index 52) have been removed. In contrast, Figure 34 depicts a state stream in which both of these components are available.

5. Cursor Types

Several different types of cursors are used within the PSDL Editor. The default cursor is the left pointer. It is used for selecting components as described above. Within the DRAWING WINDOW, an I-Bar cursor is displayed when the cursor is over

Figure 39. Hidden Components

an object (i.e., data stream or operator). The I-Bar cursor over the data flow diagram is used to indicate to the user that a label can be typed in directly. Finally, a clock face cursor is used to indicate that the PSDL Editor is busy. The clock face cursor is typically encountered when a syntax check or a prototype save function is performed. More will be said regarding the I-Bar cursor in the Data Flow Diagram paragraph below.

6. Mouse Interface

The PSDL Editor requires a two button mouse. The left-mouse button is used to access most functions. The right-mouse button is used to pull up the Stream Property or Operator Property pop-up window, as required. This is accomplished by placing the cursor over the object in question within the DRAWING WINDOW and depressing the right-mouse button. Again, more will be said about mouse interaction in the Data Flow Diagram paragraph below.

Table XIV. PSDL Editor Hot Keys

Function	Menu	Control-key	Diamond-key
Go to Root	PSDL	CTRL-R	◇-R
Go to Parent	PSDL	CTRL-P	◇-P
Decompose	PSDL	CTRL-D	◇-D
Refresh Display	Edit	CTRL-F	◇-F

7. Hot Keys

Access to selected menu functions is available through Hot Keys. Those menu items with Hot Keys are identified by an underscore under the Hot Key letter in the pull-down menu (reference Figure 31). Table XIV lists the menu functions, along with the associated key sequence, which are defined Hot Keys.

B. PSDL MAPPING

As was discussed in Chapter III, the PSDL Editor focuses the user on one operator at a time. Figure 40, similar to Figure 21 from Chapter III, indicates the general mapping of the PSDL Editor into the prototype. The PSDL graphical user interface focuses the user on the current operator. There are two exceptions, the Types Tool provides for the specification of abstract data types, at the root level, and the Operator Properties pop-up window supports the functionality of child operators. The background checker provides for the generation of redundant code in order to complete the operator's specification construct.

C. PSDL EDITOR OPERATION

The PSDL Editor is operated through the use of Menu and Tool Bar functions and pop-up windows. Upon creating a new prototype, the user must start at the root level. Abstract data types may be entered through the Types Tool at any time. However, the specification of the hierarchical tree used to implement the prototype must be performed top-down. That is, the structure of the hierarchical

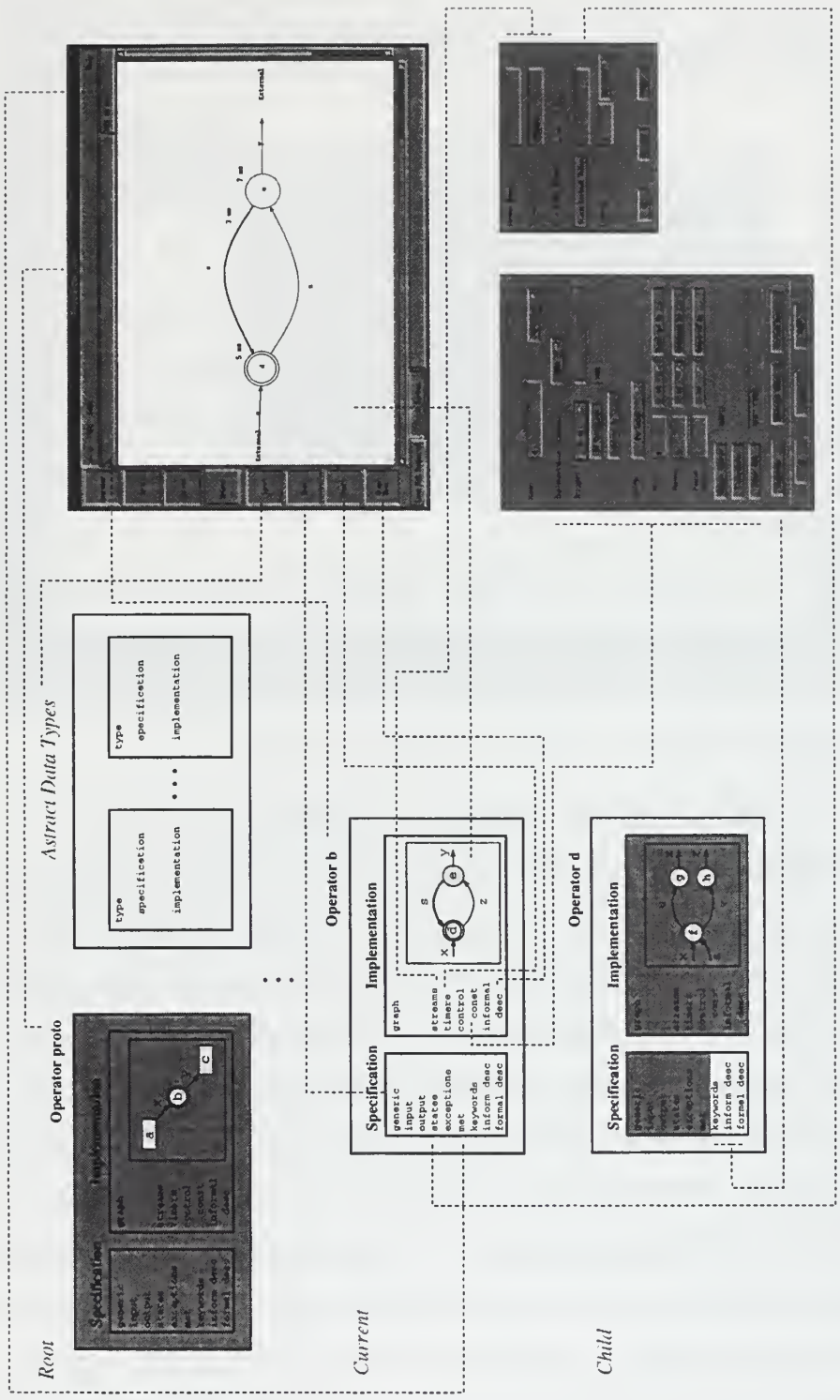


Figure 40. Editor to PSDL Mapping

tree must be specified, starting with the root and proceeding down to the leaves. Development of the prototype in this top-down fashion maintains a syntactically correct prototype. If a bottom-up approach was provided, the prototype would no longer be syntactically correct at each stage of development. The detailed implementation of each operator can be specified in any order.

The network used to implement each PSDL operator is specified within the DRAWING WINDOW, using the tools provided in the TOOL BAR. Implementation details for each composite operator and stream can be entered through pop-up properties windows. Navigation tools are provided to traverse the hierarchical tree. Any time you are using an editor, frequent saves are recommended, which may be accessed through the MENU BAR.

Finally, if help is needed, the PSDL Editor provides help windows, both at the Main Window as well as for pop-up windows. Help is accessed through a text window, as illustrated in Figure 41, with scroll bars available for browsing the message. Press the OK button to exit help.

1. PSDL Editor Segment Synchronization

The PSDL Editor consists of two segments, the background checker and the graph editor, which operate in unison. Each segment performs a portion of the editing task. The background checker is responsible for the input parsing of the PSDL prototype, extracting operators out of the prototype for processing by the graph editor, global syntax and semantic validation, and the generation of PSDL code. The graph editor is responsible for providing a graphical user interface which is used to edit/view the prototype, one operator at a time. The entire prototype is maintained within the background checker. At most one operator can be processed by the graph editor at a time. It is the responsibility of the background checker to accept the operator inputs from the graph editor and to assemble them into a prototype. User directed changes to the prototype are introduced in the graph editor. The background checker automatically resolves the implications of the graph editor produced changes

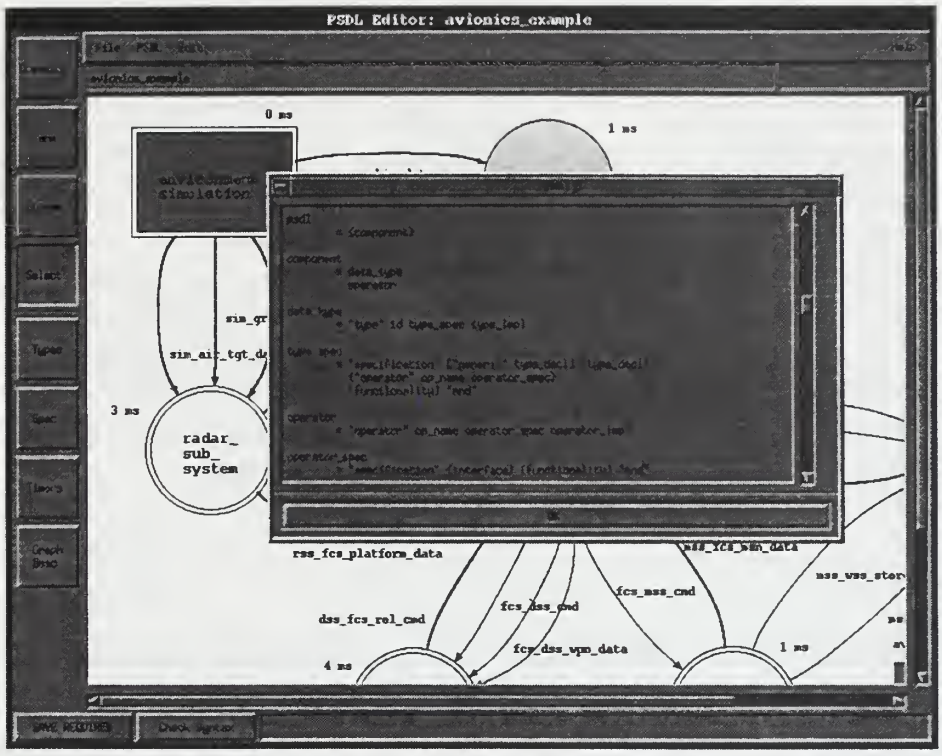


Figure 41. Help Windows

with the prototype. In the event that a discrepancy can not be resolved, an error message is fed back to the user.

The flow of information between the two PSDL Editor segments was depicted in Figure 17 of Chapter III (see page 43). After processing of the prototype, the background checker passes control, along with those objects described in `ge_interface.h` (reference Appendix D, page 195), to the graph editor. At which point the user is allowed to edit the current operator. All editing of an operator is local to the graph editor. Specific events, requested by the user, result in control being passed back to the background checker. During these events, the background checker synchronizes the prototype with respect to the graph editor. Synchronization is controlled by the `ACTION` data structure of `ge_interface.h`. The `ACTION` data structure also specifies if control is to be returned to the graph editor along with the next operator to be

Table XV. Synchronization Events and Actions

Button/Option	Located on	Action
CHECK SYNTAX button	STATUS BAR	CHECK_SYNTAX
SAVE REQUIRED button	STATUS BAR	SAVE_TO_DISK
SAVE option	FILE MENU	SAVE_TO_DISK
RESTORE FROM SAVE option	FILE MENU	REVERT
EXIT option	FILE MENU	SAVE_TO_DISK or ABANDON
SYNTAX CHECK option	PSDL MENU	CHECK_SYNTAX
GO TO ROOT option	PSDL MENU	UPDATE_TREE
GO TO PARENT option	PSDL MENU	UPDATE_TREE
DECOMPOSE option	PSDL MENU	UPDATE_TREE
ABANDON CHANGES option	EDIT MENU	ABANDON

processed.

The users requested events which result in the synchronization of the background checker and the graph editor are listed in Table XV. Associated with each synchronization event, is an action. Actions are also defined in `ge_interface.h`.

- `UPDATE_TREE` synchronizes the graph editor operator with the prototype maintained by the background checker. No validation is performed.
- `CHECK_SYNTAX` synchronizes the graph editor operator with the prototype maintained by the background checker. Syntax and semantic validation is performed.
- `SAVE_TO_DISK` synchronizes the graph editor operator with the prototype maintained by the background checker. Syntax and semantic validation is performed. The prototype is saved to a file.
- `REVERT` synchronizes the graph editor with the prototype version which resides on disk (i.e., version that was last saved).
- `ABANDON` synchronizes the graph editor with the prototype residing in the background checker memory. Only changes made since the last time the prototype was synchronized are abandoned.

2. Data Flow Diagram

The data flow diagram is specified in the DRAWING WINDOW. The PSDL Editor provides a simple drawing package, used to create/maintain the data flow

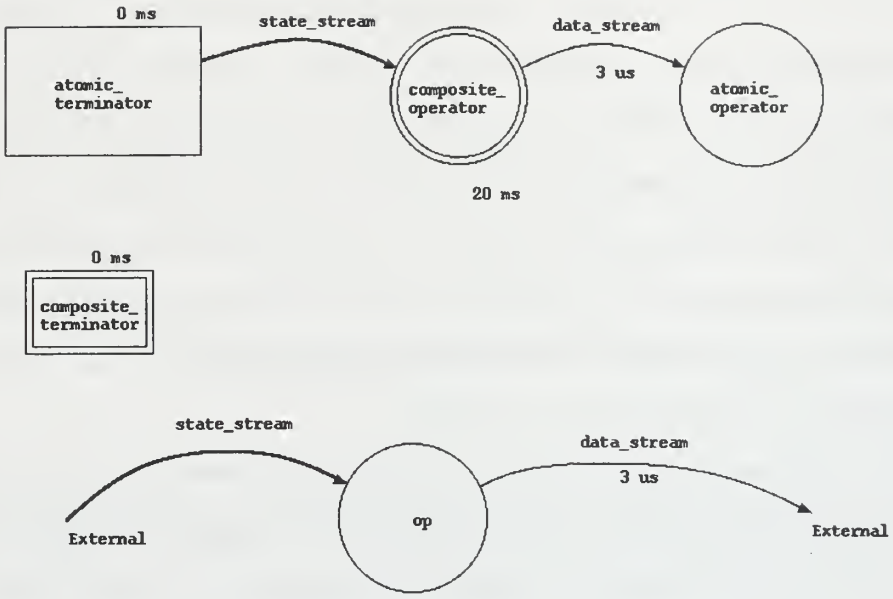


Figure 42. Data Flow Diagram Symbols

diagram. Only a small set of drawing objects are required to represent the data flow diagram. Several variations to objects, such as color and font, are provided to improve the readability of the data flow diagram. These variations have no effect on the operation of the prototype, only on the visual representation of the data flow diagram.

a. Symbols

The data flow diagram consists of a network of operators, connected through streams. Operators and streams are the only objects which are represented in the data flow diagram. Figure 42 depicts all of the symbols which are available to the user.

Chapter II introduced the distinction between operators which are considered to be part of the prototype system and those which reside outside the system. Those operators which reside outside the prototype system partition were specified by assigning a maximum execution time of zero and were called Terminators. Within

the PSDL Editor, Terminators are symbolized by rectangles. All other operators are symbolized by circles. Each PSDL Operator, including Terminators, can be implemented either as a composite operator or as an atomic operator. Atomic operators being implemented using a PSDL supported programming language, composite operators being implemented themselves by a network of PSDL operators. All composite operators are designated by a double boarder in the PSDL Editor, a double rectangle for composite terminators and a double circle for composite operators. In addition to the operator symbol, the data flow diagram is annotated with the operator name and the maximum execution time, as available.

Chapter II also introduced the distinction between streams and state streams. Streams are symbolized by a directed line, state streams by a wider directed line. In both cases, the direction of information flow is indicated by the arrow. Streams and state streams are again annotated with the stream name along with any latency value, as available.

The bottom of Figure 42 depicts the implementation of a composite operator. Streams into and out of a composite operator become external streams within the composite operator's implementation. Such streams are designated with a source or destination of `External`. Once again, the direction of information flow is indicated by the arrow.

b. Drawing

The data flow diagram is created by selecting a drawing tool from the TOOL BAR and positioning the object in the DRAWING WINDOW. The drawing tool, once selected, remains selected until another drawing tool is picked by the user. This allows the user to position as many copies of that object in the DRAWING WINDOW as desired. Specifically, the following procedures are used create the data flow diagram.

- Operators are added by first depressing and releasing the left-mouse button over the desired OPERATOR/TERMINATOR TOOL to select the tool. Then, positioning the cursor over the desired operator/terminator location and depressing and releasing the left-mouse button. Repeat for additional operators/terminators.

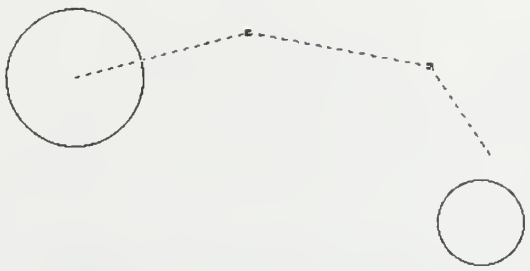


Figure 43. Construction of a Stream

- Streams are added by first depressing and releasing the left-mouse button over the STREAM TOOL to select the tool. Streams, unlike operators, are created with a sequence of points. At least two points are required to establish a source and destination. Other points can be added to route the stream around other objects on the DRAWING WINDOW. The stream source is always the first point selected. The stream destination is always the last point selected. Start the stream by positioning the cursor over the source operator and depressing and releasing the left-mouse button. Streams start and finish at the middle of an operator. There is no need to precisely position the cursor when selecting the source or destination. Route the stream using intermediate points by depressing and releasing the left-mouse button over each location. As the stream is built, it is represented with dashed line segments between anchor points. Anchor points are represented as small black squares, indicating an intermediate point. A stream under construction is depicted in Figure 43. The stream is finalized by positioning the cursor over the destination operator and depressing and releasing the left-mouse button. The resulting stream is a smooth curved line with an arrow at the destination, as depicted in Figure 44. There is no requirement to provide intermediate points within a stream. However, the PSDL Editor does not provide for the addition of intermediate points to a completed stream. Instead, the stream must be deleted and replaced. It is recommended that streams incorporate at least one intermediate point to provide for future routing changes.

- For composite operators, other than the root operator, inputs and outputs of the composite operator are represented as externals within the data flow diagram. External streams are similar to the streams above, except that they are missing either a source or destination operator. For an external source, position the cursor over the desired "source" location and depress and release the left-mouse button. Continue the stream as discussed above. For an ex-

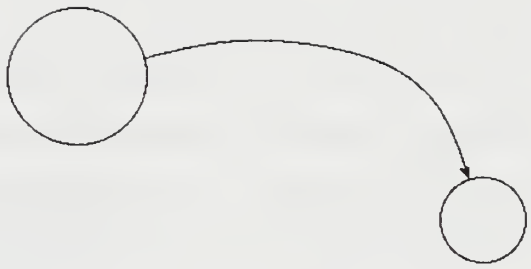


Figure 44. Completed Stream

ternal destination, start the stream as discussed above. In order to complete the stream, position the cursor over the desired "destination" location and double-depress and release the left-mouse button. Care must be taken that the mouse does not move during the double button press. Otherwise, additional intermediate points will be created. The completed external stream is as represented in Figure 42.

- It the user chooses to abort a stream while in the construction process, the Escape (Esc) can be used to cancel a stream up until completion.

- Stream and operator labels can be added directly from the DRAWING WINDOW. Position the cursor over the object which is to be labeled. Once over the object, the left pointer cursor will be replaced with an I-Bar cursor. At this point, the user can type the label. The label must correspond to a valid identifier for the object. If the cursor is moved during the process of typing the label, the PSDL Editor will assume that you are restarting the label and erase the what has been entered.

- At any time, depressing and releasing the right-mouse button over an object will access the appropriate pop-up window. After entering the desired data into the pop-up window, depress the OK to accept the data.

- In order to create a composite operator, the operator must first be selected. This is accomplished by positioning the cursor over the SELECT TOOL and depressing and releasing the left-mouse button. Next, position the cursor over the desired operator and depress and release the left-mouse button to select. The data flow diagram for the composite operator can then be accessed by selecting the DECOMPOSE option from the PSDL MENU.

- Several of the PSDL Editor functions require the selection of an object. An object is selected through the use of the SELECT TOOL, which is activated by

Figure 45. Selected Operator

depressing and releasing the left-mouse button while the cursor is located over the SELECT TOOL. Objects can then be selected by locating the cursor over the desired object and depressing and releasing the left-mouse button. As an indication that an object has been selected, the anchor points for that object are displayed, as illustrated in Figure 45.

- In order to deselect an object, depress and release the left-mouse button over any white space in the DRAWING WINDOW.

- Both operators and streams can be deleted from the data flow diagram. The procedure is the same for both operators and streams. Using the selection procedure described above, select the desired object. The object can then be deleted by the Delete key from the keyboard. If an operator is deleted, then all associated streams will also be deleted. If a stream is deleted, only that stream will be deleted.

- The PSDL Editor provides the facility to undelete an operator. The undelete operator feature is selected from the EDIT MENU. Selecting this feature access a pop-up window which contains the names of all deleted operators. Undelete the desired operator by locating the cursor over the operator name and double-depress the left-mouse button. This operation can be somewhat confusing in the event that un-named operators are deleted. Such operators show up in the names of deleted operators as a blank lines. However, the same procedure still applies. When an operator is recovered, all associated streams are also recovered. Recovery of deleted operators is available from the time the operator is deleted up until the time that the graph editor passes control back to the background checker. Once control is passed back to the background checker, all deleted operators are purged. If deleted operators are being maintained by the PSDL Editor when control is passed back to the background checker, the user is required to acknowledge that the deleted operators, together with the corresponding portions of the hierarchical tree which resides under the deleted

operators, will be purged. The user can select OK to purge or the user can remain in the graph editor by selecting NO or CANCEL.

- The PSDL Editor also provides the facility to abandon all the changes made to the PSDL operator since the last time the operator was synchronized with the prototype. This feature is accessed under the EDIT MENU with the ABANDON CHANGES option.

c. Graph Modifications

Once the data flow diagram has been defined, there are several aspects of the diagram which can be modified to improve the readability of the diagram. These changes have no effect on the performance of the prototype, only on the visual representation of the data flow diagram.

- Operators can be moved within the DRAWING WINDOW. The movement of graphics objects can only be accomplished when the SELECT TOOL is active, as described in the previous section. Using the SELECT TOOL, position the cursor over the desired operator. Depress and hold the left-mouse button. With the left-mouse button held, move the cursor to the desired location and release the left-mouse button. Labels will be moved with respect to the new operator location. Streams paths will be altered as one of the end points is relocated with the operator. None of the intermediate anchor points associated with a stream are moved.

- Streams paths can also be changed in a similar manner. Select the desired stream, as previously described, to access the stream anchor points. Still using the STREAM TOOL, position the cursor over the desired anchor point and depress and hold the left-mouse button. With the left-mouse button held, move the cursor to the desired location and release the left-mouse button. The location of stream labels will be effected by the new location of the stream mid-point.

- The size of an operator can likewise be changed through the use of anchor points. Select the desired operator using the SELECT TOOL. Position the cursor over the anchor point, which is in the direction in which you wish to change the operator's size, and depress and hold the left-mouse button. With the left-mouse button held, move the cursor to the desired location and release the left-mouse button.

- All labels, both names and time values, can be relocated on the DRAWING WINDOW. The label must be selected as previously discussed. Depress and hold the left-mouse button with the cursor over the desired label. Position the

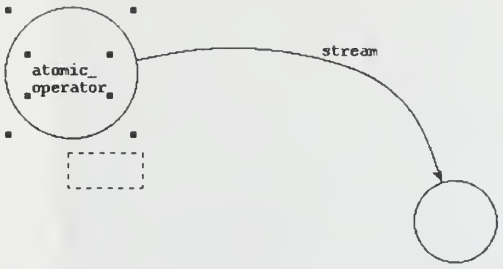


Figure 46. Relocating Operator Label

cursor over the desired location and release the left-mouse button, as depicted in Figure 46.

- If, in attempting to reposition a stream intermediate point or an operator, one of the associated labels is moved instead, make sure that the label is deselected prior to attempting to move the cursor. This may involve deselecting and reselecting the object.

- As an aid to increase the readability of the data flow diagram, operators can be colored. Selecting COLOR from the EDIT MENU access a color pop-up window. This function can be utilized in two ways. First, the color of a specific operator can be changed by selecting the desired operator and then assigning a color from the color pop-up window. Second, the color to be used for future operators can be defined by deselecting all operators before accessing the color pop-up window and then selecting a color.

- Fonts can likewise be changed from the FONT option of the EDIT MENU. Selecting the FONT menu option access a font pop-up window very similar to the color pop-up window. The same two methods of operation for COLOR apply to FONT. First, the font of a specific label can be changed by selecting the label prior to accessing the font pop-up window. Second, the font to be used for future labels can be defined by deselecting all objects before accessing the font pop-up window and then selecting a font. Note that this second mode of operation can not be used to change the font of an existing label by typing over the label with a new font. Instead, the font must be changed using the first method of operation.

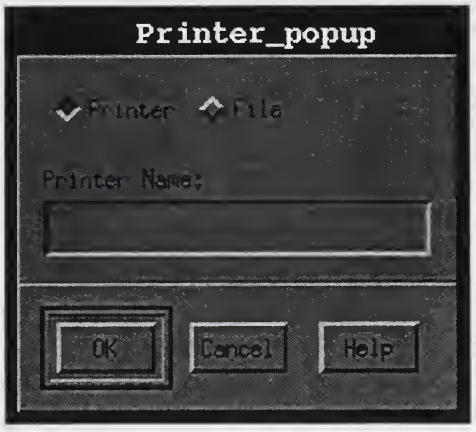


Figure 47. Printer Pop-up Window

d. Printing the Data Flow Diagram

The current data flow diagram can be saved as an output image. This capability is accessed through the PRINT option of the FILE MENU, which presents a printer pop-up window (reference Figure 47). From the printer pop-up window, the image can be saved either to the printer or a file. If file is selected, the user can provide an optional printer name. Only the printer name should be provided in the data window. If no printer name is provided, the image is printed to the standard `lpr` printer. The printer must be a PostScript[26] printer. If an output file is selected, the data flow diagram is saved to a file using `xwd`, which produces an X Window System Dump File format. The file name must be provided.

3. Navigation

The graph editor is only capable of displaying/editing the data flow diagram of one PSDL operator at a time. Each prototype consists of a hierarchical tree of PSDL operators. The PSDL Editor provides facilities to traverse the hierarchical tree in order that the entire prototype can be viewed/edited. The mechanism to traverse the prototype is provided within the PSDL MENU. The options DECOMPOSE and GO

[26]PostScript is a trademark of Adobe Systems Incorporated

TO PARENT are used to traverse up and down the hierarchical tree. Which branch of the tree is traversed, when going "down" the tree, is controlled by the operator which is selected in the data flow diagram. There is only one parent, and thus no option when traveling "up" the tree. A short cut, GO TO ROOT, is provided to directly traverse to the root operator. All navigation menu items are equipped with a hot key to improve efficiency. Hot keys were identified in Table XIV.

4. File Operations

The PSDL Editor provides very simple file operations. Since the PSDL Editor is designed to work within the CAPS environment, there are no provisions for creating a new prototype or saving the current prototype under a different name. The PSDL Editor provides two basic functions, which are located under the FILE MENU: SAVE and RESTORE FROM SAVE. A short cut button, labeled SAVE REQUIRED, is provided on the STATUS BAR. This button is context sensitive to the status of the prototype. If the prototype has not been modified, the button is labeled SAVE NOT REQUIRED. SAVE does just that, saves the prototype to disk. RESTORE FROM SAVE abandons all changes made to the prototype since the last save operation and reverts back to the last saved version. The user will be required to acknowledge that all changes will be lost. Upon completion of the restore operation, the user is returned to the root operator. The save operation will return the user to the original PSDL operator.

The user exits the editor from the FILE MENU using the EXIT option. If required, the user will be prompted to save the prototype. In addition, the user will be informed that all deleted operators will be purged.

Instead of abandoning all changes made to the prototype since the last save command, the user can abandon the changes made to an operator since the last time the operator was synchronized with the prototype. The ABANDON CHANGES option is available under the EDIT MENU for this purpose.

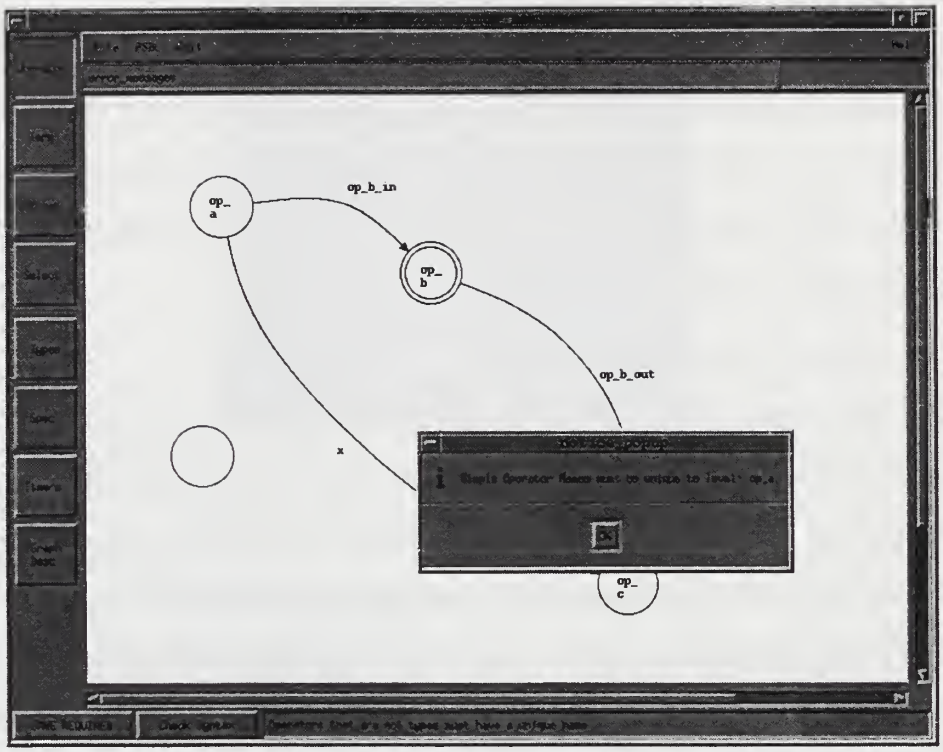


Figure 48. Graph Editor Detected Error

5. Syntax/Semantics Checking

Syntax and semantic validation tests are distributed throughout the PSDL Editor. Most syntax errors and and a few semantic errors are detected by the graph editor. The graph editor performs syntax and semantic validation upon entry of each data object, typically before the data object is written to the editor's internal representation of the prototype. Errors detected by the graph editor are reported with an information pop-up window, which must be acknowledged. An error indication is also provided in the Status Message Window, reference Component 9 in Figure 32, which serves as a reminder after the pop-up window has been acknowledged.

Figure 48 is an example of an error detected by the graph editor. In this example, the operator name op_a is being reused for the new operator in the same data flow diagram. Since op_a has already been used within the composite operator,

Table XVI. Invoking Syntax/Semantic Validation

CHECK SYNTAX button from STATUS BAR
SAVE REQUIRED button from STATUS BAR
SAVE option from FILE MENU
RESTORE FROM SAVE option from FILE MENU
SYNTAX CHECK option from PSDL MENU
ABANDON CHANGES option from EDIT MENU

an error message is generated. Upon acknowledging the pop-up notice, the Status Message Window will be retained until the next user operation. In this instance, the PSDL Editor automatically removes the duplicate name from the operator, in order to maintain a valid PSDL prototype. Had the error occurred during entry of the operator name from the Operator Property pop-up window, the invalid operator name would not have been removed. Since changes to the operator from the Operator Property pop-up window will not be accepted until all errors have ben resolved, the name is retained so that the user can correct the error.

Other validation checks can only be performed from a global perspective. These errors can only be detected by the background checker. While the graph editor performs validation upon data entry, the background checker can only validate the prototype when the prototype is synchronized with the graph editor and the background checker is passed control.

a. Invoking Syntax/Semantic Validation

Syntax and semantic validation by the background checker is performed whenever possible. In order to improve the responsiveness of the PSDL Editor, validation is not performed every time that the editor is synchronized. Table XVI identifies all user action which invokes the background checker to perform validation checks.

b. Error Messages

Upon returning command to the graph editor, any errors detected by the background checker will result in the CHECK SYNTAX button being replaced with the ERROR MSGS button on the Status Bar. By selecting this button, an Error

Messages pop-up window will be presented, as depicted in Figure 27. Within the Error Messages pop-up window, the user can select an error message and directly go to one of two operators associated with the error. The Current operator is the operator in which the background checker detected the error. The Parent operator is relative to the operator in which the error was detected.

6. PSDL Output

The output of the PSDL Editor is the PSDL code, which implements the prototype. An example of PSDL code is provided in Appendix B. Identifiers in the PSDL code differ from those observed in the PSDL Editor, with the addition of suffixes in the PSDL code. Hard copies of an operator's data flow diagram can be obtained from the PSDL Editor through the use of the PRINT function under the FILE MENU (reference Figure 31).

V. IMPLEMENTATION

The PSDL Editor has been a team effort. The architecture of having two programs (i.e., the background checker and the graph editor) work together to provide an integrated editor facility immediately lends itself to a team development. Professor Man-Tak Shing was responsible for this version of the background checker. The focus of this research has been the graph editor along with the integration of the two programs. Reference Appendix D for a listing of the complete graph editor source code. The source code for the background checker is not available in this document. However, it is available from the CAPS Research Team at the Naval Postgraduate School, Computer Science department. Appendix E provides guidelines for the installation of the PSDL Editor.

A. ARCHITECTURE OVERVIEW

The PSDL Editor architecture is characterized as having two programs, executed in separate processes, which share information through interprocess channels. The background checker provides the facilities for all file processing, input parsing of a PSDL prototype, syntax and semantic validation of the prototype, and PSDL code generation. The graph editor provides a graphical user interface and facilitates the viewing and editing of the PSDL prototype. Localized syntax and semantic validation is also performed by the graph editor.

1. Program Evolution

The current architecture of the PSDL Editor is the result of an evolution process. The CAPS Release 1 PSDL Editor was the baseline from which this research was initiated. Much of the new design is the creation of Professor Man-Tak Shing, who utilized the PSDL Editor as the class project in re-engineering of a software engineering tool. The results of this class project were then expanded upon to produce this research effort.

a. CAPS Release 1 PSDL Editor

The CAPS Release 1 PSDL Editor was developed by Captain Robert Dixon, USMC [Ref. 13]. Captain Dixon's version of the editor was implemented as two programs, the syntax-directed editor and the graph editor. Data was shared between the two programs using two files. Figure 49 illustrates the architecture used to implement the PSDL Editor. The two files supported the limited data sharing required for the data flow diagram. The first file, `gedatatransfile.txt`, was used to import the data flow diagram into the graph editor as well as to make intermediate saves. The second file, `gedatatransfile2.txt`, was used to export the data flow diagram back to the syntax-directed editor. A special file, `gedatatransfile.lock`, was used to protect the files in case multiple copies of the PSDL Editor were executed in the same directory space.

Within the graph editor, the C++ class `GraphObjectList` was used to represent the data flow diagram. Captain Dixon's thesis [Ref. 13], describes the class structure used by the graph editor. The data flow diagram itself is represented as a linked list of operator and stream objects. The common features of both of these objects are grouped together in the base class `GraphObject`, from which the `OperatorObject` and `StreamObject` classes are derived. This class structure is depicted inside the graph editor process of Figure 49.

Upon execution of the graph editor, the representation of the data flow diagram was read from `gedatatransfile.txt` into the `GraphObjectList` data structure by `build_from_disk()`. Upon returning to the syntax-directed editor, `write_to_disk()` was used to write the `GraphObjectList` data structure into the file `gedatatransfile2.txt`, to be read by the syntax-directed editor.

A command line argument was used by the syntax-directed editor to indicate if the graph editor was to be executed as a data flow diagram viewer or an editor. The same program was used for both applications.

Most of the code used to facilitate the data flow diagram was used

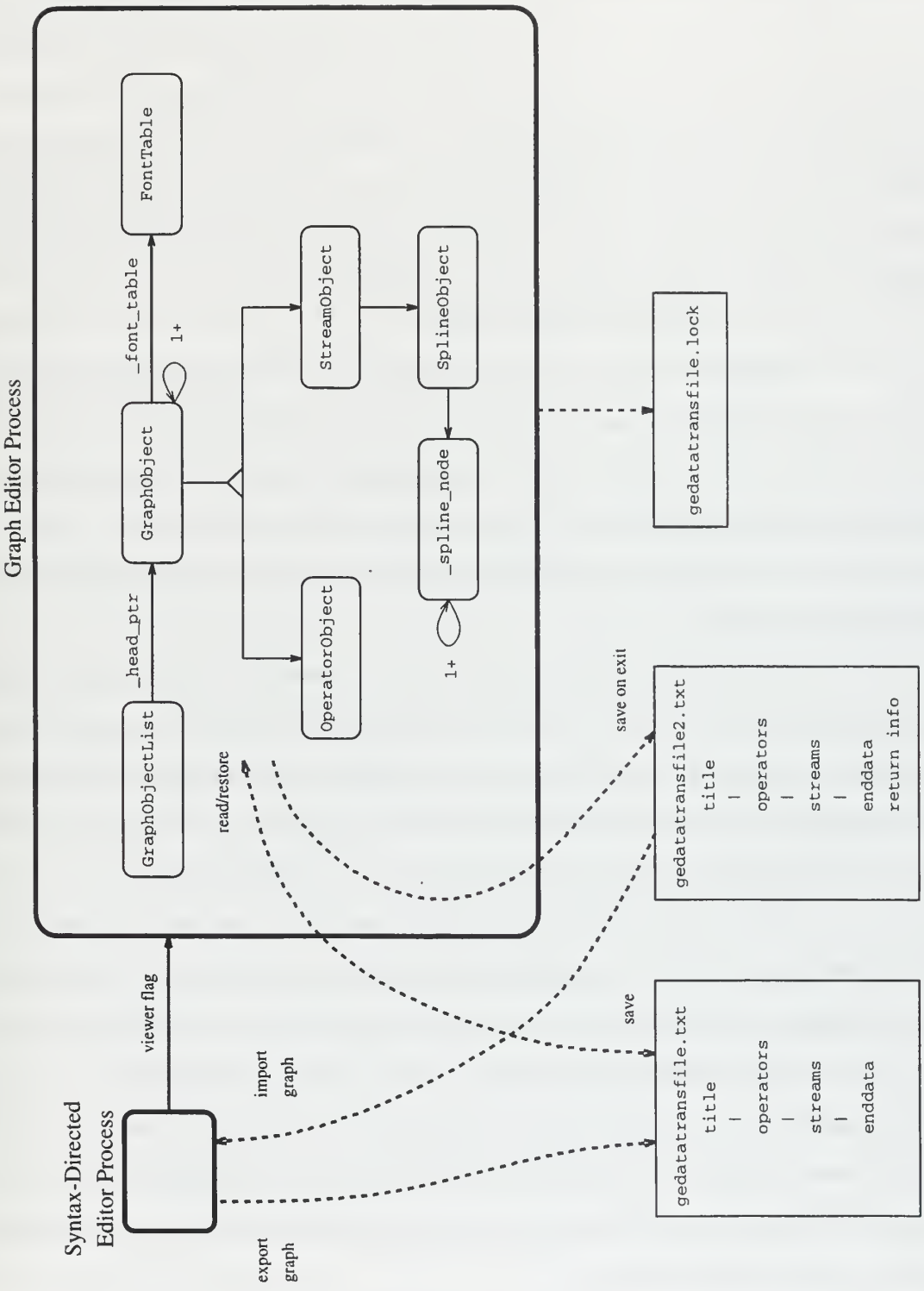


Figure 49. CAPS Release 1 PSDL Editor Architecture

directly to support this research effort. Enhancements were made to provide for the direct entry of operator and stream labels. In addition, an abort was added to the stream creation process. The two files used to share information between the syntax-directed editor and the graph editor were eliminated, along with the associated methods which supported the reading and writing of data to these files. The class structure was maintained. Additional fields and methods were added to the classes in order to support the full specification of a PSDL operator.

b. CS4520 Class Project

Professor Man-Tak Shing introduced the PSDL Editor as the class project for the Naval Postgraduate School course CS4520 (AY96Q4). CS4520 is a topical software engineering class. This particular class dealt with software re-engineering. The goal of the class project was to re-engineer the PSDL Editor to provide a more user-friendly editing facility. Only the graph editor was evaluated as part of the class project.

In addition to implementing a new user interface, the new graph editor was to be implemented as a C++ function. All data to be shared between the background checker and the graph editor was to be passed as arguments in the function call. Although only the graph editor was being implemented, the new graph editor was to be integrated with the syntax-directed editor in a single process.

The interface which was used to pass data between the background checker and the graph editor was defined by `ge_interface.h`. Data structures defined in `ge_interface.h` expanded upon the data transfered by `gedatatransfile.txt`, used in the CAPS Release 1 PSDL Editor. Within the graph editor, the original `GraphObjectList` class structure was maintained. Methods were developed to transfer data between the `ge_interface.h` data structures and the `GraphObjectList` class structure. These methods replaced the `build_from_disk()` and `write_to_disk()` methods used in CAPS Release 1.

Upon completion of the CS4520 class projects, Professors Valdis Berzins and Man-Tak Shing reviewed all the projects. The project produced by Mr. Douglas

Lange and Mr. Dagohoy Anunciado was chosen to be used for continued research of the CAPS PSDL Editor.

c. Thesis Research

Picking up from the work initiated by Mr. Douglas Lange and Mr. Dagohoy Anunciado, the completion of the graphical user interface was accomplished for the graph editor, along with implementing addition features. Initially, all work on the graph editor was accomplished in a stand-alone fashion, using a simulated background checker driver program. Later, the graph editor was integrated with the background checker. The background checker was created by Professor Man-Tak Shing, using the Synthesizer Generator. The result was an Ada/C program, which was used to call the C++ routine that was the graph editor.

Figure 50 provides a visualization of the PSDL Editor architecture. The PSDL Editor consisted of a single process. The background checker would read the specified PSDL prototype from a file and parse the prototype into the prototype data structure. The current operator would be extracted from the syntax-directed editor data structure and formatted in accordance with the `ge_interface.h` specification. This data would be passed as arguments to the graph editor function. Inside the graph editor routine, the current operator was loaded from the `ge_interface.h` data structures into the `GraphObjectList` class structure (as depicted inside the graph editor process of Figure 49). Upon exiting the graph editor, the process was reversed. The `GraphObjectList` class structure was dumped into the `ge_interface.h` data structures, which were dumped into the prototype data structurefor processing.

While development of the graph editor continued, testing was conducted as an integrated unit. X Window System problems were encountered while executing the background checker and the graph editor within a single process. These problems were not observed while executing the graph editor with a simulated background checker driver program.

Additional complications were encountered while troubleshooting the system. With the background checker being implemented in Ada and C and the

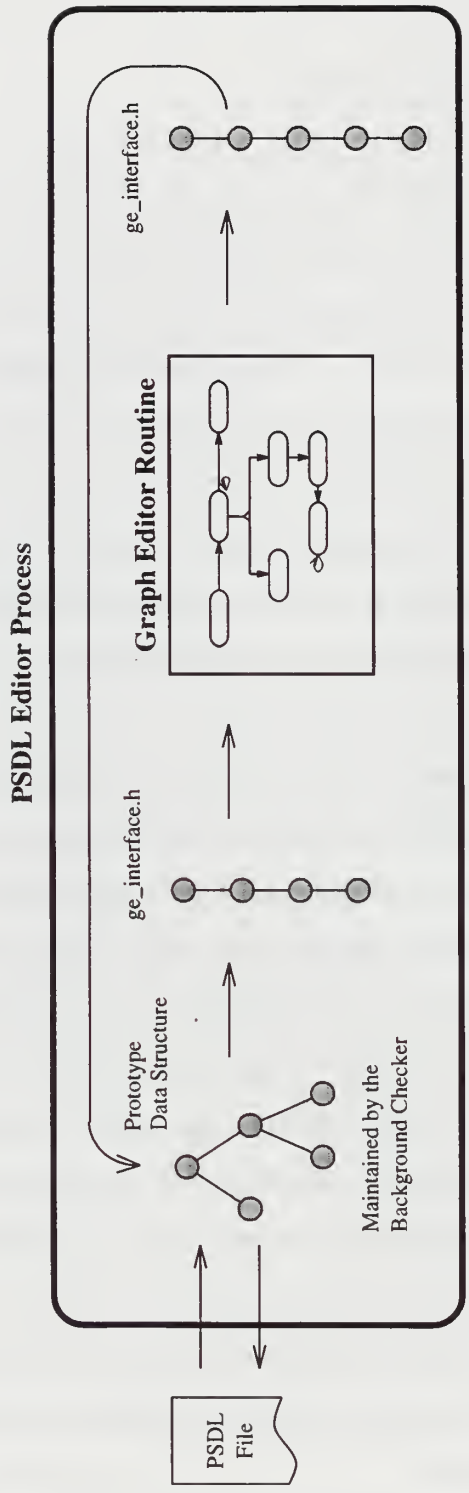


Figure 50. Initial PSDL Editor Architecture

graph editor being implemented in C++, difficulties were encounter using the source debuggers. Finally, errors reported within the X Window System routines could not be traced since there was no source code for the X Window System. This configuration resulted in difficulty troubleshooting all problems with the editor, typically using output statements to debug the program.

These problems were resolved by once again splitting the PSDL Editor into two segments, the background checker and the graph editor, executed in two separate processes. This configuration also supported the source debuggers, which were a great aid to the development of the PSDL Editor.

Figure 51 provides a visualization of the final PSDL Editor architecture, in which the graph editor is executed in a separate process. The background checker is responsible for forking a child process and establishing the pipe lines for interprocess communication. Within the background checker, the graph editor routine has been replaced with calls to transfer the `ge_interface.h` data structures to the graph editor process. A driver `ge_driver.C` routine was added to the front of the graph editor. This routine is responsible for accepting the `ge_interface.h` data structures over the interprocess communication pipe line and passing them onto the graph editor routine. Upon exiting the graph editor routine, the `ge_driver.C` routine passes the `ge_interface.h` data structure back to the background checker.

Early testing of the PSDL Editor was accomplished using a small prototype. This prototype was filled with PSDL features, which tested most of the PSDL Editor, however, it was not a very stressful test. Later, additional testing was accomplished using a medium size prototype. Upon testing with the `avionics_example` prototype, found in Appendix B, problems were encountered. The initial problem, which was obvious to any user, was the delays encountered while using the PSDL Editor. Where one of the initial complaints of the CAPS Release 1 PSDL Editor was the delays between interacting with the syntax-directed editor and the graph editor, the new implementation had even longer delays. Delays from 30 to 45 seconds were

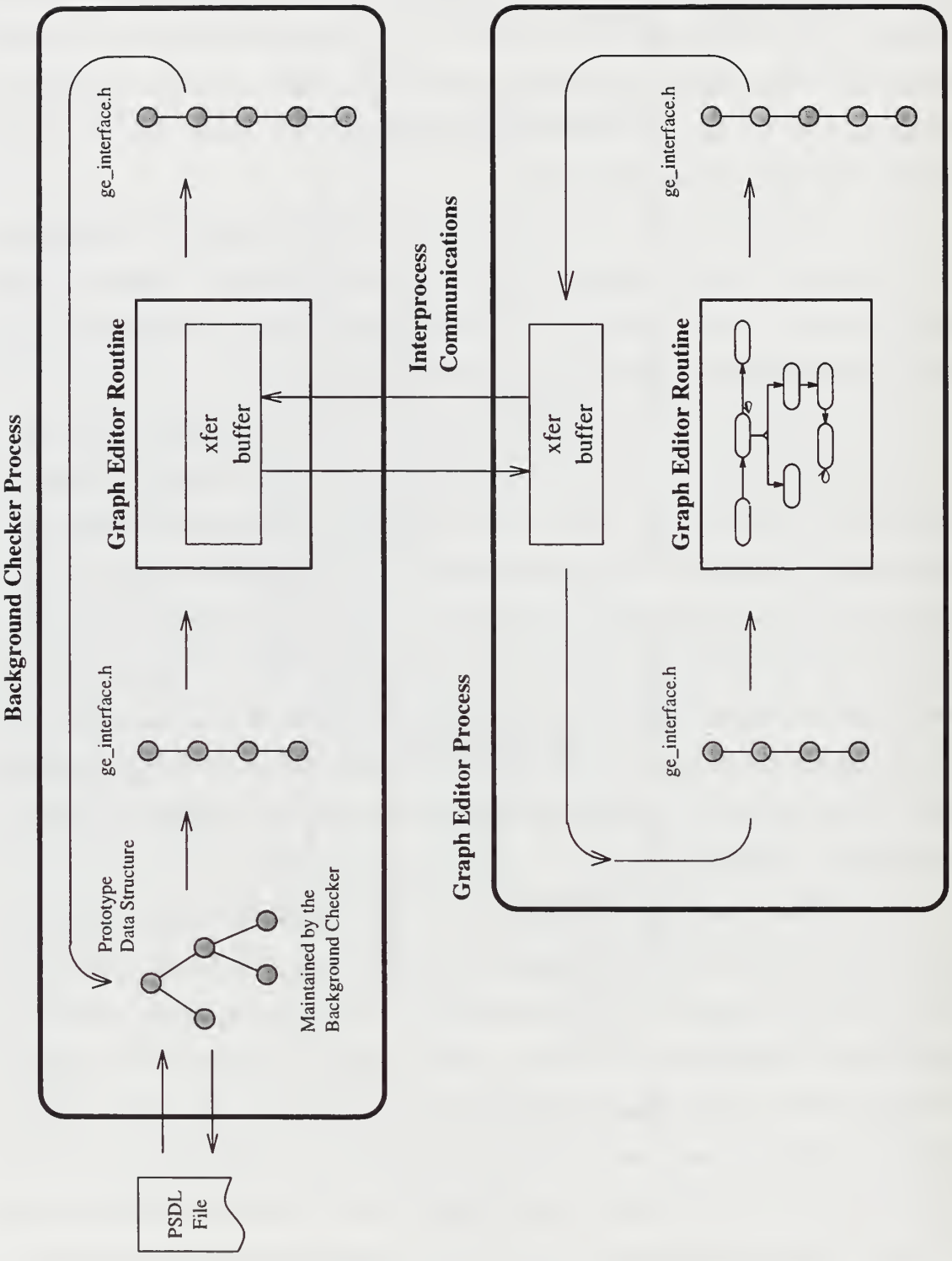


Figure 51. Final PSDL Editor Architecture

typical when navigating through the avionics_example.

Prolonged use of the PSDL Editor resulted in the second big problem, the system ran out of memory. In such a case, the PSDL Editor session had to be aborted. All unsaved changes to the prototype were lost. These problems are further defined under the Lessons Learned paragraph of Appendix B.

Analysis of the PSDL Editor indicated that both the delay and the memory problems were due to the background checker. These problems, as well as the desire to increase portability by removing the reliance on the Synthesizer Generator, resulted in the development of the new Ada background checker.

2. Architecture

The PSDL Editor, in its final configuration, consists of two segments, the background checker and the graph editor. As depicted in Figure 51, each segment is executed within its own process. Communication between the two segments is accomplished using Unix pipes. Two pipes are opened to provide bi-directional communication between the two processes. Input to the PSDL Editor consists of a single file containing the PSDL prototype. Output of the PSDL Editor is also a single PSDL file. All user interaction is performed through the graphical user interface provided by the graph editor. The graphical user interface is implemented as an X Window System application using the Motif widget set. Communication between the two processes is based upon the ge_interface.h data structures.

3. Data Communications

The interface between the background checker and the graph editor supports the bi-directional transfer of PSDL data. In addition, errors detected by the background checker are transfered to the graph editor. While the graph editor commands the background checker to perform the next action. All interprocess communication is defined by the file ge_interface.h. Along with the data structures to support the above communications, ge_interface.h provides common definitions.

a. Graph Description

The structure GRAPH_DESC provides all of the information regarding the prototype required by the graph editor. Only one operator is processed by the graph editor at a time, although some global prototype information is shared. The GRAPH_DESC structure represents the operator's data flow diagram as a linked list of OPERATOR structures as well as a linked list of STREAM structures. The GRAPH_DESC structure is used in bi-directional communication with the background checker and the graph editor.

b. Error Messages

Errors detected by the background checker can only be presented to the user through the graphical user interface provided by the graph editor. The data structure ERROR_MSGS is a linked list of error messages, along with associated operator and parent operator identification. If no errors are detected by the background checker, a NULL pointer shall be provided to the graph editor.

c. Next Action

The data structure ACTION provides the background checker with the commands required to determine the next action to take. Instruction is provided on how to synchronize the data contained within GRAPH_DESC with the prototype data structure, maintained by the background checker. Also provided are instructions on returning to the graph editor with the desired current operator.

4. Synchronization

Within the graph editor, all changes to an operator are local to the graph editor. Local changes are synchronized with the prototype maintained by the background checker upon specific user requested events. The synchronization, along with the user specified event, were listed in Table XV of Chapter IV. All synchronization is a function of the background checker. A distinction was made between UPDATE_TREE and CHECK_SYNTAX as a concession to performance. Initially, syntax validation was performed with each synchronization event. However, large delays to perform syn-

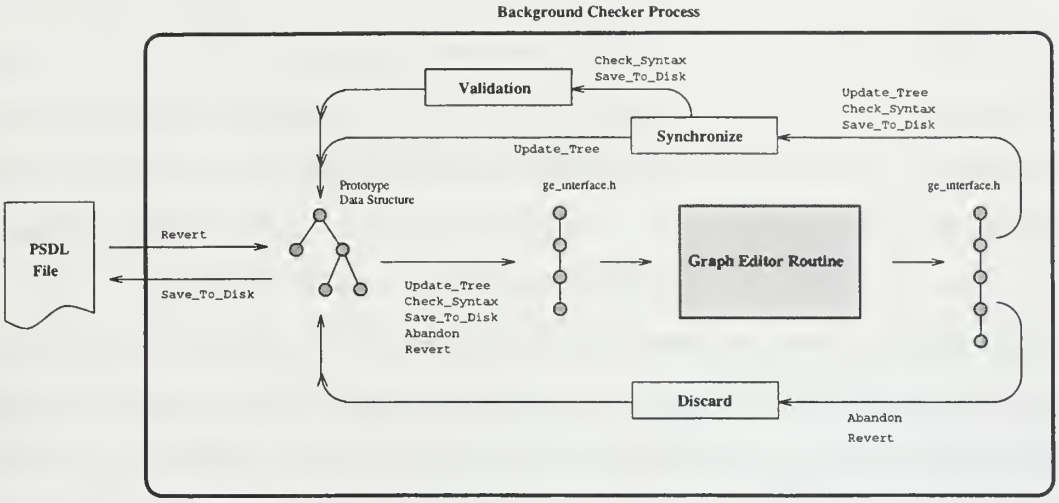


Figure 52. Background Checker Synchronization

tax validation made this option unattractive for all synchronization events. Thus, `UPDATE_TREE` was added as an option, in which the prototype data structure, maintained by the background checker, were synchronized with any modifications made in the graph editor, but with no syntax validation. This option was provided to improver performance while navigating the prototype. Figure 52 illustrates the type of processing which is expected to be performed by the background checker based on the synchronization command.

B. GRAPH EDITOR DATA STRUCTURES

Within the graph editor, the `GraphObjectList` class structure is used to maintain all PSDL data. Captain Dixon described this structure in his thesis [Ref. 13]. This class structure has been maintained with this release of the PSDL Editor. A class diagram of the `GraphObjectList` structure was provided within the graph editor process box of Figure 49.

All information contained within `GRAPH_DESC`, from `ge_interface.h`, which does not relate to the data flow diagram was appended to the `GraphObjectList`

class. Changes were made to all classes that incorporated a time literal. In the CAPS Release 1 PSDL Editor, time literals were encoded as signed integers. All time values in microseconds were encoded as negative times. Values in milliseconds were encoded as positive times. In order to support all time units defined in the CAPS Release 2 PSDL grammar, time literals were maintained with two symbols. One maintained the time value, the other maintained the time units.

Data flow diagram representation information was appended to class objects `OperatorObject` and `StreamObject` as applicable. Label and time values, displayed on the data flow diagram, for both operators and streams were modified to function as offsets from the graphics object. Previously, labels and time values were recorded at absolute locations. Methods were added to support the `ge_interface.h` structures.

C. UTILITIES

Several sets of utilities files were created in the process of developing the PSDL Editor.

1. Graph Editor Utilities

The files `ge_utilities.h` and `ge_utilities.c` provide a set of routines for dealing with components of the PSDL Editor. These files were written in C in order to support both the graph editor as well as the original syntax-directed editor (which was written in C due to the Synthesizer Generator). Primarily, the routines in these files support the maintenance of the data structures defined in `ge_interface.h`. The routine `dup_str()` is used throughout the PSDL Editor. This routine is used to make a deep copy of a string. In general, the PSDL Editor is implemented using deep copies. Shared values have been avoided.

Another facility provided by the `ge_utilities` routines is the validation of PSDL constructs. These routines are used by the graph editor to validate identifiers, operator identifiers, type names, keywords, and integer literals.

Table XVII. Inter-Process Communication Routines

`readAction()`	`writeAction()`
`readErrorMsgs()`	`writeErrorMsgs()`
`readGraphDesc()`	`writeGraphDesc()`

2. Inter-Process Communication

The inter-process communication package was designed to accept the data structures defined by `ge_interface.h` at one end and recreate the same data structures on the other end. These communication facilities are provided by the routines in the files `inter_process_utilities.h` and `inter_process_utilities.c`. Once again, these files were written in C to support both the background checker and the graph editor.

The inter-process communication facility is provided by six routines listed in Table XVII. These routines operate on a `xfer_buffer`. The `xfer_buffer` is allowed to expand, doubling in size each time, in order to support the largest data structure. The `xfer_buffer` itself is sent over Unix pipes in packets with a maximum size of 4096 bytes.

The pipe lines used to facilitate inter-process communications are established by the background checker upon creation of the graph editor process. Two pipe lines are opened, in order to provide bi-directional communication. The two channels to be used are passed to the graph editor as command line arguments. The file `ge_driver.C` is the `main()` routine for the graph editor. This file processes the command line arguments containing the channel numbers. The code used to open the pipe lines is provided by the background checker. A sample of code used to facilitate this process may be found in the file `sde.c`, which is discussed below.

3. Unique Identifier Generator

The incorporation of unique suffixes for operators required a unique suffix generator. The files `get_unique_id.h` and `get_unique_id.c` provide this facility.

Once again, these files were developed in C in order to support both the graph editor and the background checker. This implementation uses a file named `unique_id.dat` from which to generate sequential integers. Future releases of the PSDL Editor shall utilize a data base to provide unique identifiers across a distributed development environment (reference Professor Berzins' memo included as Attachment C). The current implementation is limited to the visibility of the `unique_id.data` file.

4. Program Development Aids

Several files used develop the PSDL Editor have been included within the source code for the graph editor. While not required for operation of the PSDL Editor, they may be useful for future development efforts.

a. Driver Program

The files `sde.c`, `sde_simulator.h`, and `sde_simulator.c` were developed to perform stand alone testing of the graph editor. The file `sde.c` provides the `main()` routine used to fork the graph editor in its own process as well as to establish communications. The `sde_simulator` files provide a hard-coded prototype which can be edited. Note that this driver program has not been designed to work interactively with the graph editor. The driver program does not contain the facilities to synchronize with the graph editor.

While the source code for the background checker is not contained here, the file `sde.c` does provide an example of code used to fork the graph editor process and to establish communication pipe lines.

b. Debug Routines

In the process of integrating the background checker and the graph editor, often there was a need to examine the data contained within the `ge_interface.h` data structures. A collection of routines provided in files `ge_utilities_debug.h`, `ge_utilities_debug.c`, and `ge_interface_labels.h` were developed to display and print to a file the contents of the `ge_interface.h` data structures.

D. GRAPH EDITOR

The files used to implement the graph editor, roughly organized by function, are listed in Table XVIII. The graph editor routine which actually edits the prototype is `edit_graph()`, defined in `graph_editor.C`. This routine is called from `ge_driver.C` upon receiving the `ge_interface.h` data structure from the background checker. At every synchronization event, this routine is exited. Within `edit_graph()`, the Motif windows are initialized. Initialization is performed once, based on the symbol `motif_initialized`. After the initial entry, the Motif environment is assumed to exist. Motif data is maintained within global objects in order to provide persistence.

Typically, a Motif application expects to pass control over to Motif until the program terminates. In order to support the PSDL Editor, the Motif control loop had to be maintained within the graph editor. The `edit_graph()` routine enters a loop which dispatches X Window System events until a synchronization event is selected by the user to exit the graph editor.

Table XVIII. Graph Editor Source Code Files

Function	Header File	Code File
Graph Editor	graph_editor.h	ge_driver.c graph_editor.C
Definition Files	ge_interface.h ge_defs.h resources.h	
Classes	graph_object_list.h graph_object.h operator_object.h stream_object.h spline_object.h font_table.h	graph_object_list.C graph_object.C operator_object.C stream_object.C spline_object.C font_table.C
Pop-up Windows	operator_property_menu.h stream_property_menu.h	operator_property_menu.C stream_property_menu.C
Window Utilities	action_area.h build_option.h gettopshell.h postpopup.h report_errors.h setcursor.h timer_tool.h warning.h windows.h	action_area.C build_option.c gettopshell.c postpopup.c report_errors.C setcursor.c timer_tool.C warning.C windows.C
Utilities	ge_utilities.h inter_process_utilities.h get_unique_id.h sde_simulator.h ge_utilities_debug.h ge_interface_labels.h	ge_utilities.c inter_process_utilities.c get_unique_id.c sde.c sde_simulator.c ge_utilities_debug.c

VI. CONCLUSIONS AND RECOMMENDATIONS

Overall, this research was successful in the development of an improved PSDL Editor facility. While full graphical user interface support was not provided for all features of PSDL, sufficient portions of the language were supported to demonstrate the concepts of the improved interface.

Since the aspects of the PSDL Editor addressed by this research primarily relate to human factors, any validation of the improvements made would have to be accomplished with a survey of users. Such a survey was not conducted as part of this research, so no conclusions can be stated regarding the effectiveness of the improvements. However, a few personal observations from experience obtained while testing the PSDL Editor can be made along with some recommendations for future research.

A. RESULTS OF RESEARCH

Based on personal experience with the PSDL Editor, I would submit that the artificial boundary between the syntax-directed editor and the graph editor has been significantly reduced. What remains of the boundary is the batch mode of operation when it comes to global syntax and semantic validation as well as any delays encountered switching between the two programs. While early versions of the improved PSDL Editor had significant delays, optimizations within the background checker have resulted in acceptable delay times, even for large prototypes.

Use of pop-up windows to specify the details of an operator or stream has streamlined the development of prototypes. Immediate access to complete details of a data flow diagram object are now one button away. No longer are users compelled to complete the data flow diagram prior to entering any of the details, due to the long delays and steps required to access this data. Direct entry of operator and stream

labels has been a significant step in streamlining the development process.

Previous problems encountered by users attempting to locate a construct within the syntax-directed editor should not be as prevalent with the use of pop-up windows. These problems, associated with a linear translation of the PSDL grammar, are reduced by the depiction of all options within the pop-up window. Options displayed in the pop-up window are consistent with the current configuration of the object. No longer must a user remember the ordering of PSDL constructs, they are visible to the user from the pop-up window. Ghosting of an option acts to remind the user of the availability of an option while the physical location of the options suggest dependencies between options.

An improvement which should reduce the amount of lost work in the PSDL Editor was the simplification of file operations. All file save operations now write directly to the prototype file. There is no longer an intermediate file to which the editor saves the prototype, from which users have lost entire edit sessions due to a misunderstanding of the save procedure. The currently implemented file system should be intuitive to most users of personnel computer software.

As previously mentioned, not all features of PSDL were implemented with the graphical user interface. Several features of PSDL were determined to be too complex for this initial implementation of the graphical user interface. These features included abstract data types, constraint options, expressions, and the specification interface. Although not supported by the graphical user interface, these features were not left unsupported. Text windows were provided within the graph editor for the specification of these constructs using PSDL. Being some of the more complex PSDL constructs, they are also some of the more advanced features. And while all these features are critical to the use of PSDL, they are most often used by more advanced users. The result is that the PSDL Editor provides sufficient capabilities to be utilized for prototype development by the average user. More advanced users can still utilize the PSDL Editor, however, they will require more knowledge of PSDL to access these

advanced features of PSDL.

B. CRITIC OF RESEARCH

As with any program, criticism can be made of the design and implementation. This program offers no exception. Discussed below are observations of problems in the design and implementation. I do not point a finger to any previous author of the PSDL Editor for problems with this release. All of the problems, both in design and implementation, were mine to address. What few problems were inherited from previous authors were most likely small problems which I have amplified in "improving" the program. But just as there was insufficient time to support all features of PSDL, there was insufficient time to address these "features" of the PSDL Editor.

1. User Interface

The mapping between the PSDL Editor's graphical user interface and PSDL is not clean. For instance, the functionality of an operator's specification is accessed through the pop-up window for a composite operator. This mapping does not support the root operator.

The drawing operation of the data flow diagram is too slow. While there may be a tendency to coerce the user to maintain good programming practices and keep the data flow diagram simple, it should not be accomplished by aggravating the user with delays. The drawing of streams, and especially state streams, is extremely slow. The system is not usable, when executed remotely over phone lines.

Associated with the delays involved with drawing the data flow diagram, the data flow diagram is too sensitive to change. Unintentional changes are typically made to the data flow diagram as the user attempts to navigate the prototype. In order to navigate through the prototype, the user must select an operator. If the user moves the mouse while the left-mouse button is depressed, the object will be moved and the user will have to wait for the data flow diagram drawing to be updated.

The pop-up facilities to modify an operator's color or a label's font as well as those used to recover a deleted operator are rudimentary. No provisions are made to indicate the currently selected color or font. No provisions are made to Ok the selection or to cancel the operation. The user is required to know how to select the item with a double mouse action as well as the current value, in case the user wishes to leave the value unchanged. No feedback is provided to the user to indicate that a color or font change is being performed on a specific object or the default value is being changed. The restoration of an unlabeled operator is a guessing game for the user.

2. Implementation

The PSDL Editor started out as two programs, executed in separate processes. As an update to the design, it was converted over to a single program, executed in a single process. Problems encountered with this configuration forced it to be split once more into two programs, executed in separate processes. Again, problems were encountered which forced a change to the design of the editor. This last change has seen the removal of the Synthesizer Generator produced background checker, to be replaced with an Ada version. With this latest change, no modification was made to the interface between the background checker and the graph editor. With the design changes which have occurred so far, maintaining two separate programs may be wise. However, at this point, two separate processes appears to be unnecessary. Associated with two separate processes is a lot of communication overhead which could be removed, thus simplifying the program. Combining the two tasks into one procedure would also improve the performance with respect to delays, thus further reducing the artificial boundary between the background checker and the graph editor.

With the evolution of the graph editor, the program has become "messy". The program has appeared to have lost any structure as new features are simply added. Global objects are used throughout the program to provide communication with the graphical user interface. Files have become much too large. The `graph_editor.C`

file, once printed, goes on for fifty pages. The `draw` routine alone is ten pages long, with far too many levels of indentation. Even simple maintenance to the program becomes complex and error prone with this absence of organization.

Within the graph editor, a clean representation of data objects has not been maintained. Redundant data is maintained throughout the data objects. For instance, which graphical object is selected is indicated within the data structure for that object. Thus a house keeping process must be maintained to ensure that only one object is selected at at time, clearing all other selected objects. The result is that, occasionally, multiple graphic objects appear to be selected. The user is confused at this point. Only by selecting and de-selecting each object that appears to be selected can the indications be cleared up.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

In the process of re-engineering the PSDL Editor, many additional ideas come to mind. Most of which are much too late to be incorporated into the design of the system. The following is a collection of ideas which should be used to stimulate thought for the next release of the PSDL Editor.

The current design of the graphical user interface provides a very limited view of the context of an operator. The user is presented with the data flow diagram of the operator, but no visualization of the context of the operator. The CAPS Release 1 PSDL Editor provided some indication of external streams. Even this limited view of context was removed from this release of the PSDL Editor. Future releases of the PSDL Editor should prompt the user with options available from the operator's context. Figures 53 and 54 depict an initial attempt to provide the user with such a context with respect to streams. Figure 53 illustrates the user's options of entering a stream name directly into the data entry window or to select from a list of predefined streams. Figure 54 depicts all streams which are currently defined in the context, along with an indication of external streams and their use, state streams and their

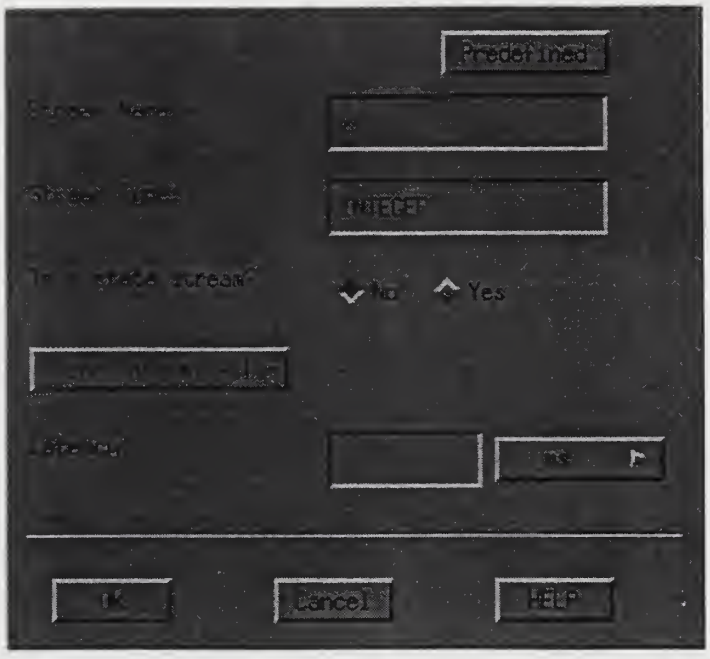


Figure 53. Stream Property Pop-up with Predefined Option

Figure 54. Stream Predefined Context

initial value, and the stream type. Due to scheduling constraints, this option was never incorporated into the final release of the PSDL Editor.

Options available for improved data flow diagram editing capabilities mostly involve modifications to streams. Such features as being able to remove or add intermediate stream anchor points as well as being able to change the direction of a stream would improve the user's ability to modify an existing data flow diagram. Higher level edit features for working with operators would also greatly improve the user's ability to modify a prototype. Such features would include the ability to select a group of operators from a single data flow diagram and form a composite operator from them, thus adding one more level to the hierarchical tree. The reverse operation could also be provided as a feature, to remove a level from the hierarchical tree. Copy, cut, and paste operations on selected operators could be provided. Modifications to object properties, such as color and font, could be performed on several selected objects simultaneously.

Improved semantic validation capabilities could be incorporated. Possible candidates include detecting cycles in data flow diagrams which are not broken by state streams and global validation of timing constraints. This release of the PSDL Editor took a step backward in editing an existing prototype. In order to edit a prototype, the editor must be able to parse the input file. Syntactical errors in the input file could result in the PSDL Editor not being able to parse the prototype. Previously, the PSDL Editor would at least allow the user to edit the prototype text if a syntactical error was detected. With this release, the PSDL Editor is not usable on a corrupted prototype.

While the CAPS environment provides access to the complete set of CAPS tools, many of these tools could be integrated with the PSDL Editor. Tighter integration of CAPS tools could provide:

- Automatic generation of skeleton files to support atomic operators. This capability has become a necessity with the inclusion of unique suffixes to operators.

- Direct access to an editor to view/edit an atomic operator implementation.
- Access to the reuse library from the PSDL Edit based on an operator's specification.
- A PSDL Editor mode for performing maintenance on the reuse library.
- The ability to save a prototype as a new revision within the Evolution Control System.
- The ability to merge prototypes.
- The ability to examine the prototype schedule from the editor. Computer assistance should be provided on how to modify the prototype in order to change the schedule.

Finally, with the advances made in internet and intranet technologies comes the opportunity to improve upon existing applications. The migration of the PSDL Editor's help facilities to a hyper-text mark-up language (HTML) based system should be straight forward. Such a system would be a great improvement over the existing capability. The conversion of the PSDL Editor over to a internet/intranet application would be much more involved. However, the resulting facility would provide a portable editor capable of supporting distributed prototype development.

APPENDIX A. PSDL GRAMMAR

The following is the complete specification of the Prototype System Description Language (PSDL) syntax in an extension of Backus-Naur Form (BNF) [Ref. 14].

The BNF description of PSDL specifies the sequence of symbols which constitute a valid PSDL prototype. BNF describes the language in terms of production rules. Each production rule equates a non-terminal symbol to a sequence of terminal and non-terminal symbols. Terminal symbols are symbols which can occur in PSDL. Non-terminal symbols are metalinguistic variables whose value is a sequence of symbols which represents a PSDL construct.

Terminals are represented as **bold** symbols (e.g., **operator**). Non-terminals are enclosed in angle brackets, ⟨ and ⟩ (e.g., ⟨*psdl*⟩). Additional metasymbols are introduced in the extension of BNF to reduce the number of productions and non-terminals. These metasymbols are defined as:

- Square brackets, [ ], enclose optional items.
- Curly braces, { }, enclose items which may appear zero or more times.
- Vertical bars, |, represent a choice between items.
- Parentheses, ( ), represent a grouping of items.

In some cases, the metasymbols used are also used as terminals within PSDL. In order to avoid confusion, such terminal symbols are enclosed within single quotes, (e.g., ')').

For ease of reference, each production rule is numbered on the left hand side. These numbers are not part of the PSDL syntax.

1 ⟨*psdl*⟩
::= { ⟨*component*⟩ }

2 ⟨*component*⟩
::= ⟨*data_type*⟩
| ⟨*operator*⟩

3 ⟨*data_type*⟩
::= **type** ⟨*id*⟩ ⟨*type_spec*⟩ ⟨*type_impl*⟩

4 ⟨*type_spec*⟩
::= **specification** [**generic** ⟨*type_decl*⟩] [⟨*type_decl*⟩]
{ **operator** ⟨*op_name*⟩ ⟨*operator_spec*⟩ }
[⟨*functionality*⟩] **end**

5 ⟨*operator*⟩
::= **operator** ⟨*op_name*⟩ ⟨*operator_spec*⟩ ⟨*operator_impl*⟩

6 ⟨*operator_spec*⟩
::= **specification** { ⟨*interface*⟩ } [⟨*functionality*⟩] **end**

7 ⟨*interface*⟩
::= ⟨*attribute*⟩ [⟨*reqmts_trace*⟩]

8 ⟨*attribute*⟩
::= **generic** ⟨*type_decl*⟩
| **input** ⟨*type_decl*⟩
| **output** ⟨*type_decl*⟩
| **states** ⟨*type_decl*⟩ **initially** ⟨*initial_expression_list*⟩
| **exceptions** ⟨*id_list*⟩
| **maximum execution time** ⟨*time*⟩

9 ⟨*type_decl*⟩
::= ⟨*id_list*⟩ : ⟨*type_name*⟩ { , ⟨*id_list*⟩ : ⟨*type_name*⟩ }

10 ⟨*type_name*⟩
::= ⟨*id*⟩
| ⟨*id*⟩ '[' ⟨*type_decl*⟩ ']'

11 ⟨*id_list*⟩
::= ⟨*id*⟩ { , ⟨*id*⟩ }

12 ⟨*reqmts_trace*⟩
::= **required by** ⟨*id_list*⟩

13 ⟨*functionality*⟩
::= [⟨*keywords*⟩] [⟨*informal_desc*⟩] [⟨*formal_desc*⟩]

14 ⟨*keywords*⟩
::= **keywords** ⟨*id_list*⟩

15 ⟨*informal_desc*⟩
::= **description** '**{**' ⟨*text*⟩ '**}**'

16 ⟨*formal_desc*⟩
::= **axioms** '**{**' ⟨*text*⟩ '**}**'

17 ⟨*type_impl*⟩
::= **implementation** ⟨*id*⟩ ⟨*id*⟩ **end**
| **implementation** ⟨*type_name*⟩
{ **operator** ⟨*op_name*⟩ ⟨*operator_impl*⟩ } **end**

18 ⟨*operator_impl*⟩
::= **implementation** ⟨*id*⟩ ⟨*id*⟩ **end**
| **implementation** ⟨*psdl_impl*⟩ **end**

19 ⟨*psdl_impl*⟩
::= ⟨*data_flow_diagram*⟩ [⟨*streams*⟩] [⟨*timers*⟩]
[⟨*control_constraints*⟩] [⟨*informal_desc*⟩]

20 ⟨*data_flow_diagram*⟩
::= **graph** { ⟨*vertex*⟩ } { ⟨*edge*⟩ }

21 ⟨*vertex*⟩
::= **vertex** ⟨*op_id*⟩ [**:** ⟨*time*⟩] { ⟨*property*⟩ }

22 ⟨*edge*⟩
::= **edge** ⟨*id*⟩ [**:** ⟨*time*⟩] ⟨*op_id*⟩ **–>** ⟨*op_id*⟩ { ⟨*property*⟩ }

23 ⟨*property*⟩
::= **property** ⟨*id*⟩ **=** ⟨*expression*⟩

24 ⟨*op_id*⟩
::= [⟨*id*⟩ **.**] ⟨*op_name*⟩ ['(' [⟨*id_list*⟩] '|' [⟨*id_list*⟩] ')']

25 ⟨*streams*⟩
::= **data stream** ⟨*type_decl*⟩

26 ⟨*timers*⟩
::= **timer** ⟨*id_list*⟩

27 ⟨*control_constraints*⟩
::= **control constraints** ⟨*constraint*⟩ { ⟨*constraint*⟩ }

28 ⟨*constraint*⟩
::= **operator** ⟨*op_id*⟩
[**triggered** [⟨*trigger*⟩] [**if** ⟨*expression*⟩] [⟨*reqmts_trace*⟩]]
[**period** ⟨*time*⟩ [⟨*reqmts_trace*⟩]]
[**finish within** ⟨*time*⟩ [⟨*reqmts_trace*⟩]]
[**minimum calling period** ⟨*time*⟩ [⟨*reqmts_trace*⟩]]
[**maximum response time** ⟨*time*⟩ [⟨*reqmts_trace*⟩]]
{ ⟨*constraint_options*⟩ }

29 ⟨*constraint_options*⟩
::= **output** ⟨*id_list*⟩ **if** ⟨*expression*⟩ [⟨*reqmts_trace*⟩]
| **exception** ⟨*id*⟩ [**if** ⟨*expression*⟩] [⟨*reqmts_trace*⟩]
| ⟨*timer_op*⟩ ⟨*id*⟩ [**if** ⟨*expression*⟩] [⟨*reqmts_trace*⟩]

30 ⟨*trigger*⟩
::= **by all** ⟨*id_list*⟩
| **by some** ⟨*id_list*⟩

31 ⟨*timer_op*⟩
::= **reset timer**
| **start timer**
| **stop timer**

32 ⟨*initial_expression_list*⟩
::= ⟨*initial_expression*⟩ { , ⟨*initial_expression*⟩ }

33 ⟨*initial_expression*⟩
::= **true**
| **false**
| ⟨*integer_literal*⟩
| ⟨*real_literal*⟩
| ⟨*string_literal*⟩

| ⟨*id*⟩
| ⟨*type_name*⟩ . ⟨*op_name*⟩ ['(' ⟨*initial_expression_list*⟩ ')']
| '(' ⟨*initial_expression*⟩ ')'
| ⟨*initial_expression*⟩ ⟨*binary_op*⟩ ⟨*initial_expression*⟩
| ⟨*unary_op*⟩ ⟨*initial_expression*⟩

34 ⟨*binary_op*⟩
::= **and** | **or** | **xor**
| < | > | = | >= | <= | / =
| + | − | & | * | / | **mod** | **rem** | **

35 ⟨*unary_op*⟩
::= **not** | **abs** | − | +

36 ⟨*time*⟩
::= ⟨*integer_literal*⟩ ⟨*unit*⟩

37 ⟨*unit*⟩
::= **microsec**
| **ms**
| **sec**
| **min**
| **hours**

38 ⟨*expression_list*⟩
::= ⟨*expression*⟩ { , ⟨*expression*⟩ }

39 ⟨*expression*⟩
::= **true**
| **false**
| ⟨*integer_literal*⟩
| ⟨*time*⟩
| ⟨*real_literal*⟩
| ⟨*string_literal*⟩
| ⟨*id*⟩
| ⟨*type_name*⟩ . ⟨*op_name*⟩ ['(' ⟨*expression_list*⟩ ')']
| '(' ⟨*expression*⟩ ')'
| ⟨*expression*⟩ ⟨*binary_op*⟩ ⟨*expression*⟩
| ⟨*unary_op*⟩ ⟨*expression*⟩

40 ⟨*op_name*⟩
::= ⟨*id*⟩

41 ⟨*id*⟩
::= ⟨*letter*⟩ { ⟨*alpha_numeric*⟩ }

42 ⟨*real_literal*⟩
::= ⟨*integer_literal*⟩ . ⟨*integer_literal*⟩

43 ⟨*integer_literal*⟩
::= ⟨*digit*⟩ { ⟨*digit*⟩ }

44 ⟨*string_literal*⟩
::= " { ⟨*char*⟩ } "

45 ⟨*char*⟩
::= *any printable character except* '**}**'

46 ⟨*digit*⟩
::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

47 ⟨*letter*⟩
::= **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m**
| **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z**
| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M**
| **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z**

48 ⟨*alpha_numeric*⟩
::= ⟨*letter*⟩
| ⟨*digit*⟩
| '_'

49 ⟨*text*⟩
::= { ⟨*char*⟩ }

APPENDIX B. PROTOTYPE EXAMPLE

Early in the development of the PSDL Editor, a small prototype created by Dr. Shing was used for testing purposes. This prototype consisted of a few operators, and hence did not present a sufficient test case for the PSDL Editor. The lack of size of Dr. Shing's prototype lead to the creation of an `avionics_example` prototype.

The goal of the `avionics_example` prototype was to use as many features of PSDL as possible and provide a medium to large size prototype to test the PSDL Editor. Since the PSDL Editor has limited facilities for types and generics, examples of these features were limited or non-existent. In addition, no attempt was made to test all combinations of expression evaluations.

In addition, no attempt was made to provide a full avionics suite. The example presented is not intended to reflect an actual avionics suite, although aspects of the prototype can be found in aircraft that are currently in the United States Air Force inventory.

1. ARCHITECTURE

Figure 55 represents the avionic suite that is modeled in the prototype. The avionic suite consists of a six subsystems: Fire Control System (FCS), Radar SubSystem (RSS), Navigation SubSystem (NSS), Weapon SubSystem (WSS), Display SubSystem (DSS), and the Mass-Storage Subsystem (MSS). The FCS contains two dual-redundant processors, FCS_1 and FCS_2. The RSS contains both a radar antenna and an inertial platform on the antenna, which can be used as a backup navigation source. Two bomb racks are provided in the WSS.

The avionic suite contains two data buses. One is used for the RSS and NSS. The other is used for the WSS, DSS and MSS. A single bus controller is used to coordinate all messages traveling over the buses. The FCS performs all bus controller activities. Since the FCS is dual-redundant, one FCS processor is designated as the

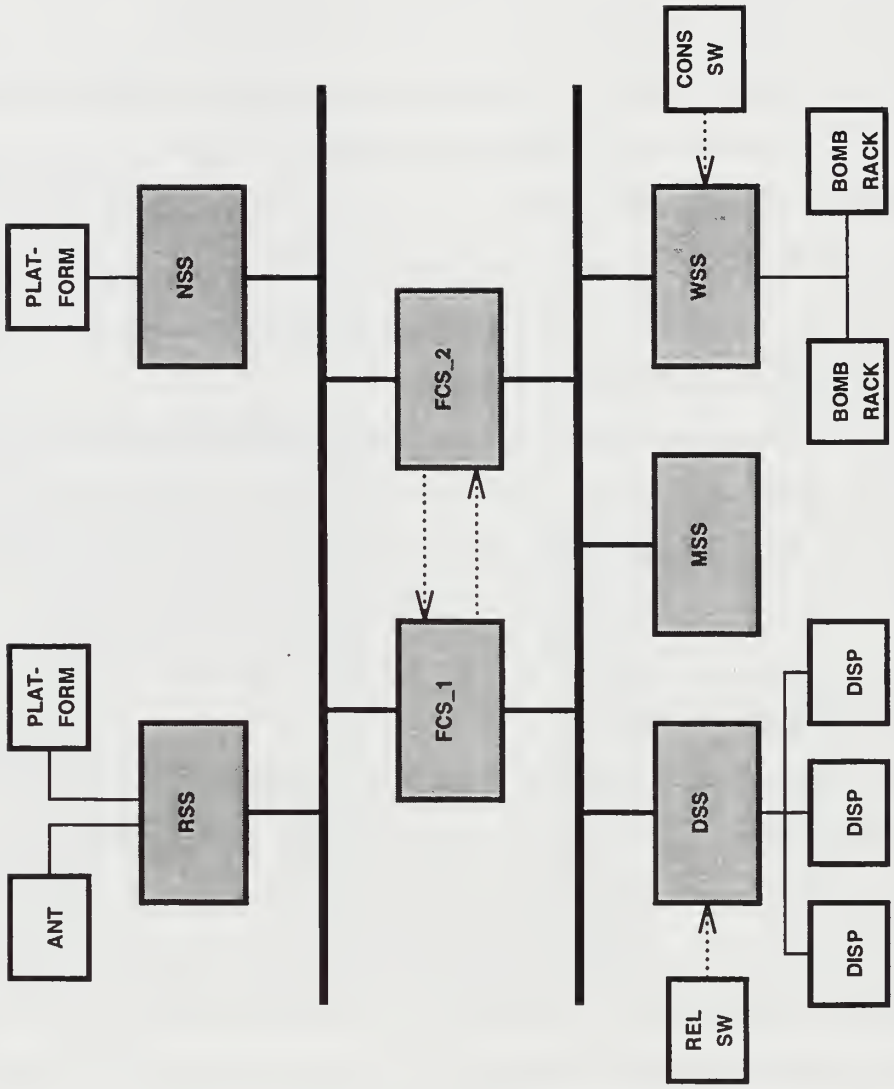


Figure 55. `avionics_example` Architecture

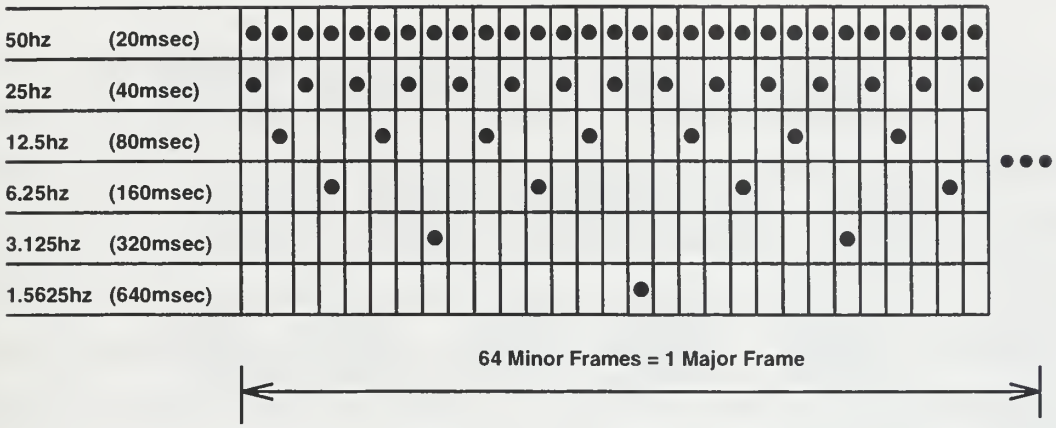


Figure 56. Minor Cycle Task Scheduling

bus controller while the other monitors the health of the bus controller. If the bus controller should fail, the second FCS processor takes over bus controller activities.

All tasks performed by the avionic suite are scheduled to be accomplished at different rates. The execution of each processor is divided into minor cycles and major cycles. For a typical avionic suite, the processors are executed at 50Hz. Thus, each minor cycle lasts for 20 milliseconds. A major cycle would consist of 64 minor cycles. All tasks are expected to be accomplished in a major cycle. During a minor cycle, tasks from two rate groups are performed. This is depicted in Figure 56 where each column represents a minor cycle. During each minor cycle, tasks from the 50Hz rate group and a slower rate group are performed.

2. MAPPING TO PSDL

Just as the design of an avionic suite would start with a high level diagram and decompose each component of the design, the prototype consists of a root operator, `avionics_example`, which is composed of a PSDL operator for each subsystem. In the case of the NSS, a simple pass-thru model will be used. No additional decomposition is required for a simple operator. Thus, the NSS is implemented in Ada. The other subsystems are more complex, and thus, are decomposed into a PSDL graph

implementation.

Although no software base is provided for this prototype to facilitate the execution of the prototype, two additional prototype components are required to complete the model. An `environment_simulation` operator is required to simulate the aircraft motion as well as the target environment and an `air_vehicle_interface` operator is required to facilitate the displays produced by the `display_sub_system` as well as providing a user interface for switch actions. Both of these operators are external to the avionic suite and are thus represented as terminators (square boxes) so that the execution time is not counted against the avionic suite.

For the purposes of this prototype, the timing model previously discussed is too complex. Instead, tasks will be accomplished at either 50Hz, 25Hz or 1Hz. Each subsystem that performs tasks at different rates contains a mode control operator which schedules the tasks.

In the actual avionic suite, each processor executes in parallel. Tasks within a processor are performed serially. While PSDL does not preclude the parallel execution of operators, CAPS Release 1 was implemented on a single processor. This prototype is also designed to be executed on a single processor. In order to maintain the desired execution rate of 50Hz, each subsystem is assigned an execution time which is a fraction of a minor cycle (reference Figure 57). Since the processing of each subsystem is simulated, the calculations performed can be greatly simplified in order meet the timing requirements.

3. PROTOTYPE

The decomposition of each PSDL operator for the `avionics_example` prototype is provided in Figures 58 through 64.

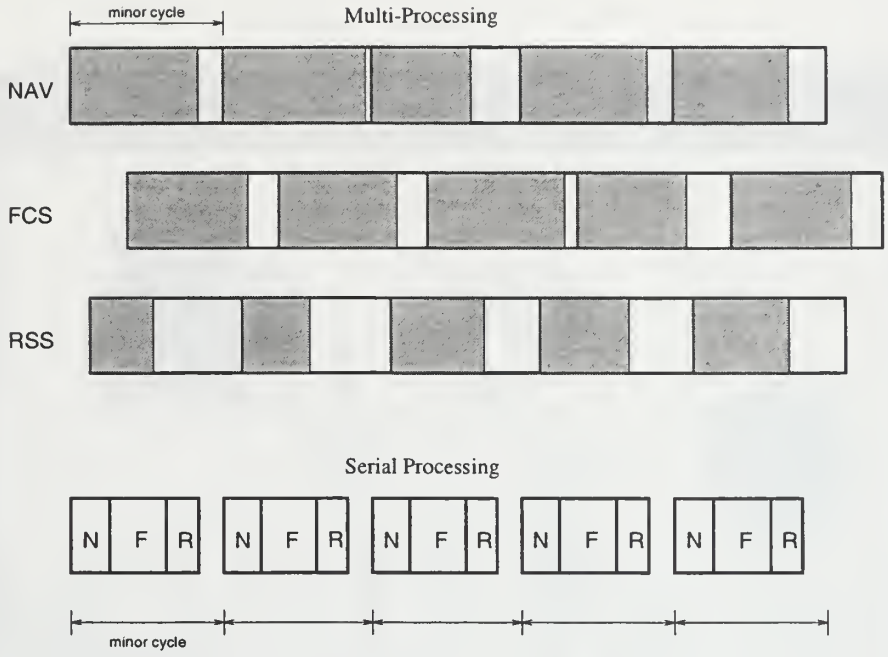


Figure 57. Serializing Multiple Processors

4. CAPS RELEASE 1 COMPATIBILITY

Although this implementation of the PSDL Editor incorporated changes to the CAPS Release 1 grammar, an attempt was made to schedule the prototype using CAPS Release 1[27]. Before the prototype can be scheduled, a successful translation must be accomplished. It was found that the translator would not accept the **property** construct added to the PSDL grammar. Nor did the translator accept an implementation language other than `Ada`[28]. The inclusion of operator and vertex identification numbers to the ⟨*op_name*⟩ also caused compatibility problems. Since the ⟨*op_name*⟩ provided in an ⟨*operator*⟩ ⟨*component*⟩ does not include the vertex identification number, where the ⟨*op_id*⟩'s used in the ⟨*data_flow_diagram*⟩ does,

[27]Note that there is no implied compatibility of this implementation of the PSDL Editor with CAPS Release 1. This experiment was intended to identify compatibility issues.

[28]The CAPS Release 1 PSDL Editor did not accept a mixed case `Ada`. An all upper case `ADA` was accepted. Additional problems encountered with the CAPS Release 1 PSDL Editor include the **finish within** construct and `Missing_Info` containing two underscores in succession

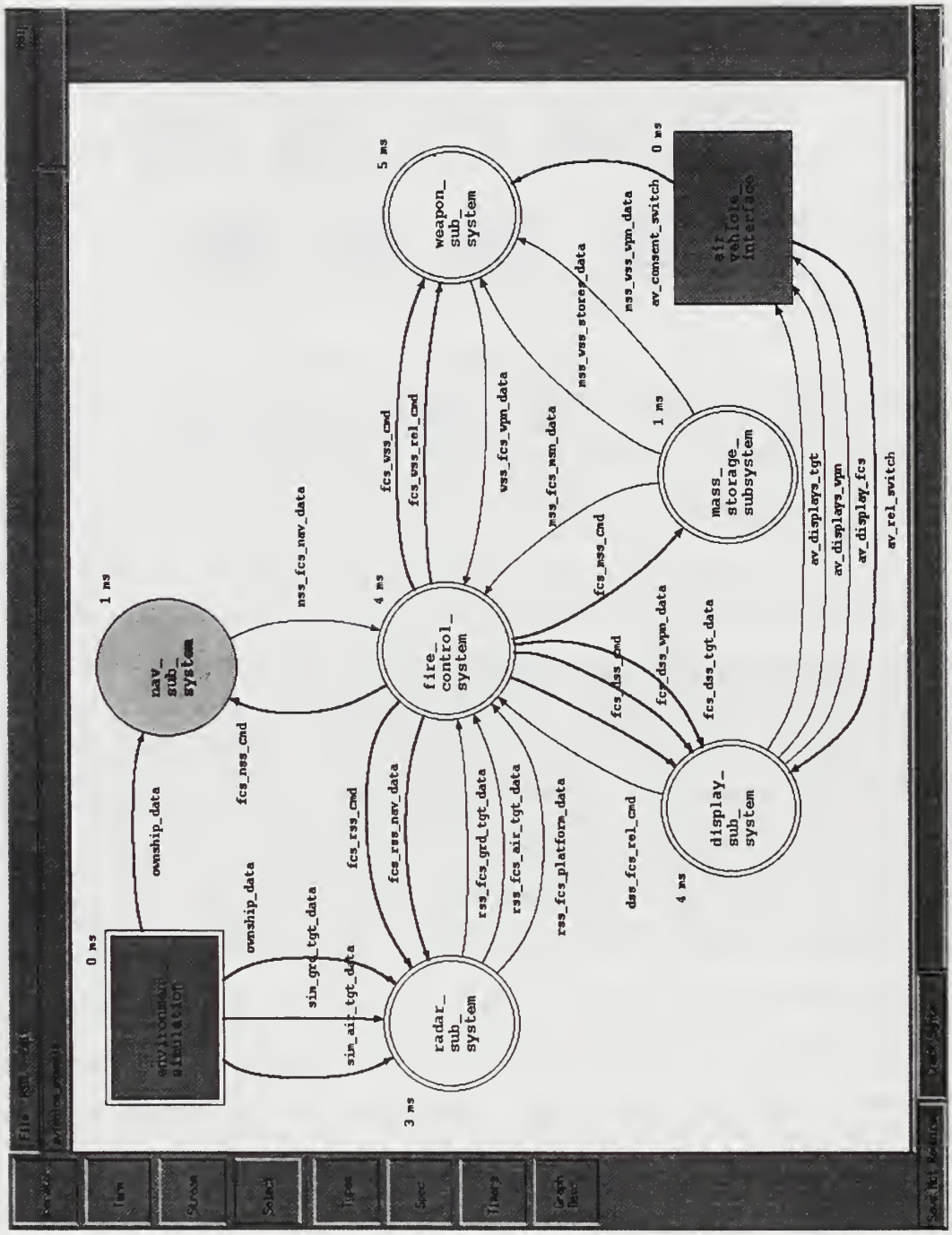


Figure 58. avionics_example Root Operator

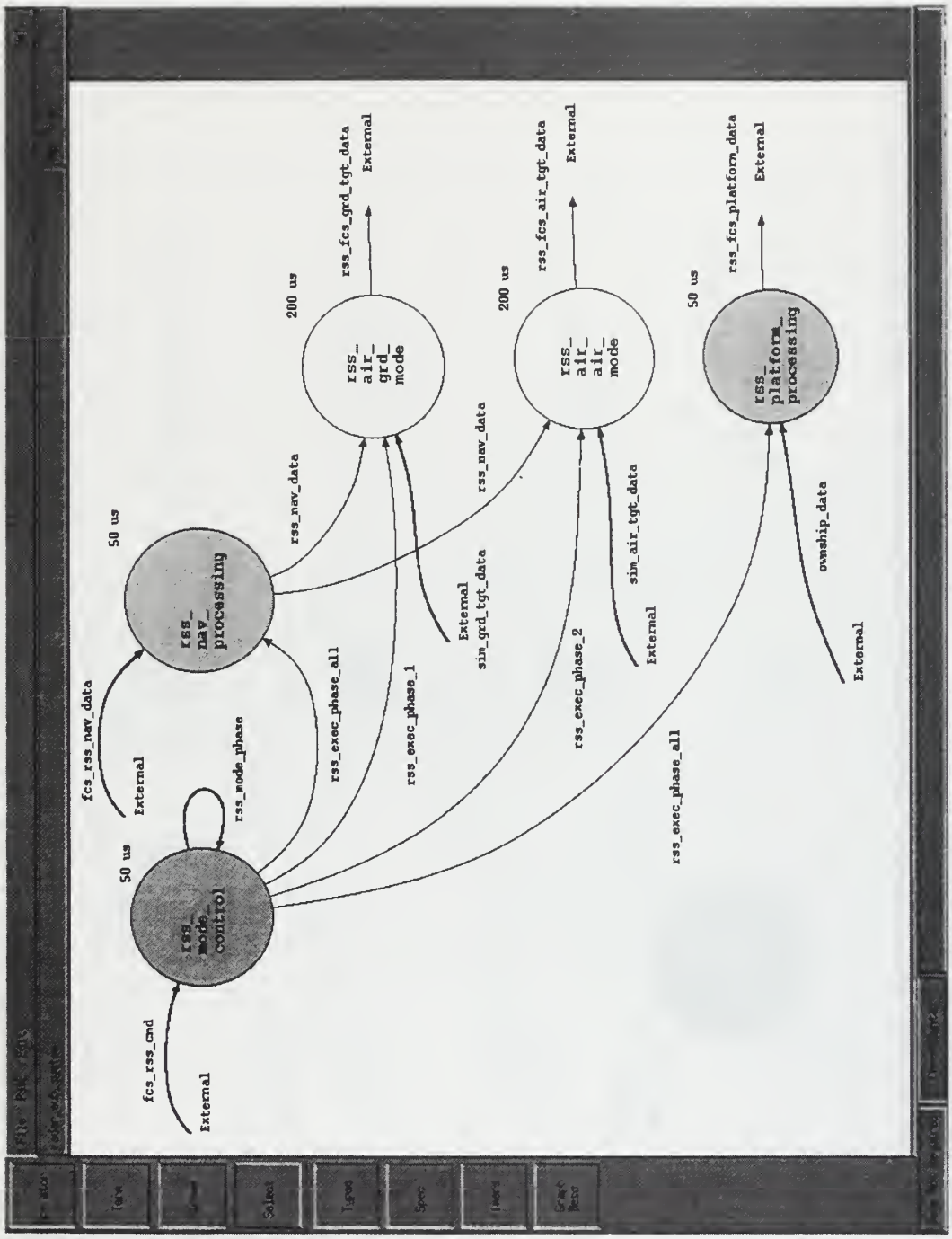


Figure 59. avionics_example RSS Operator

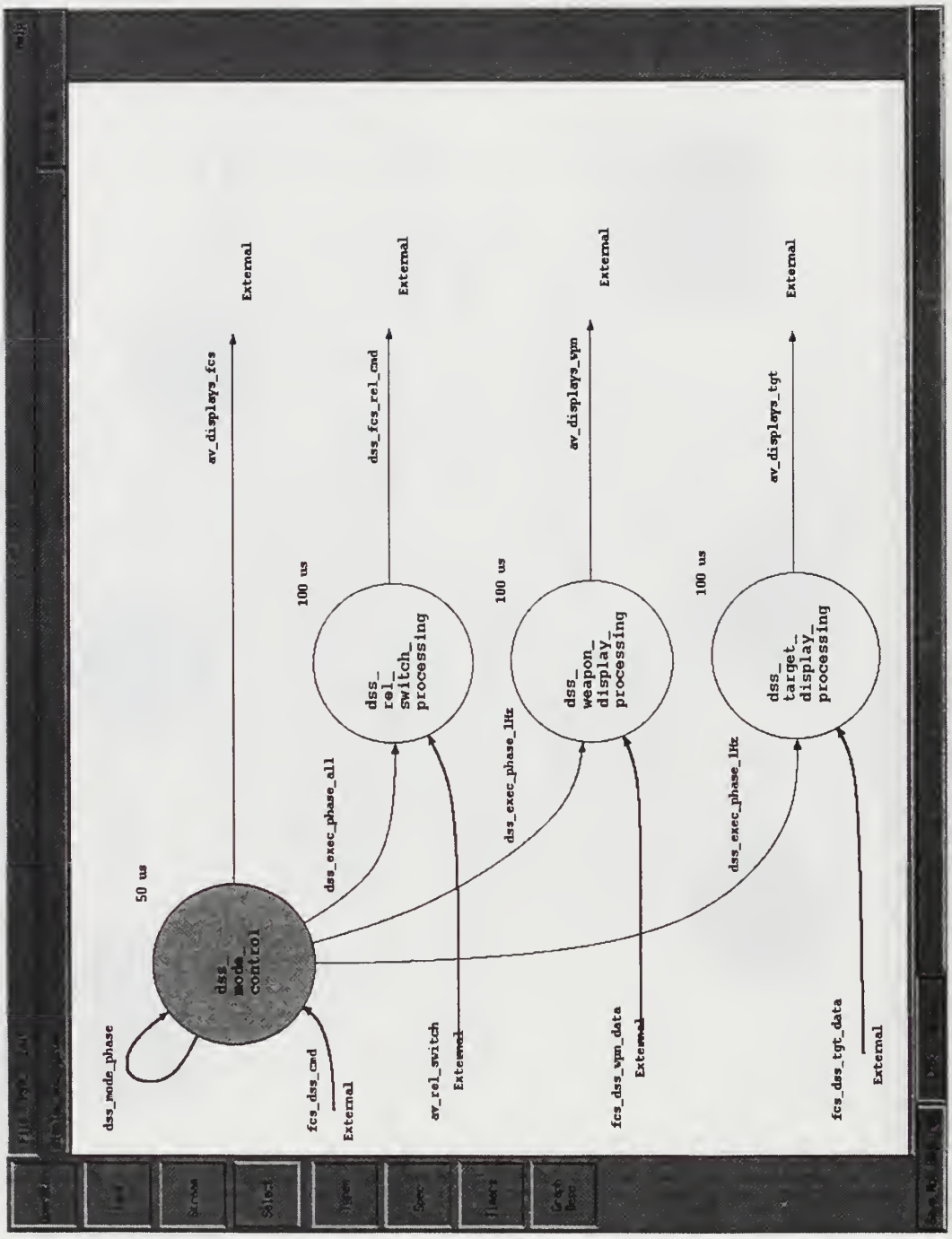


Figure 60. avionics_example DSS Operator

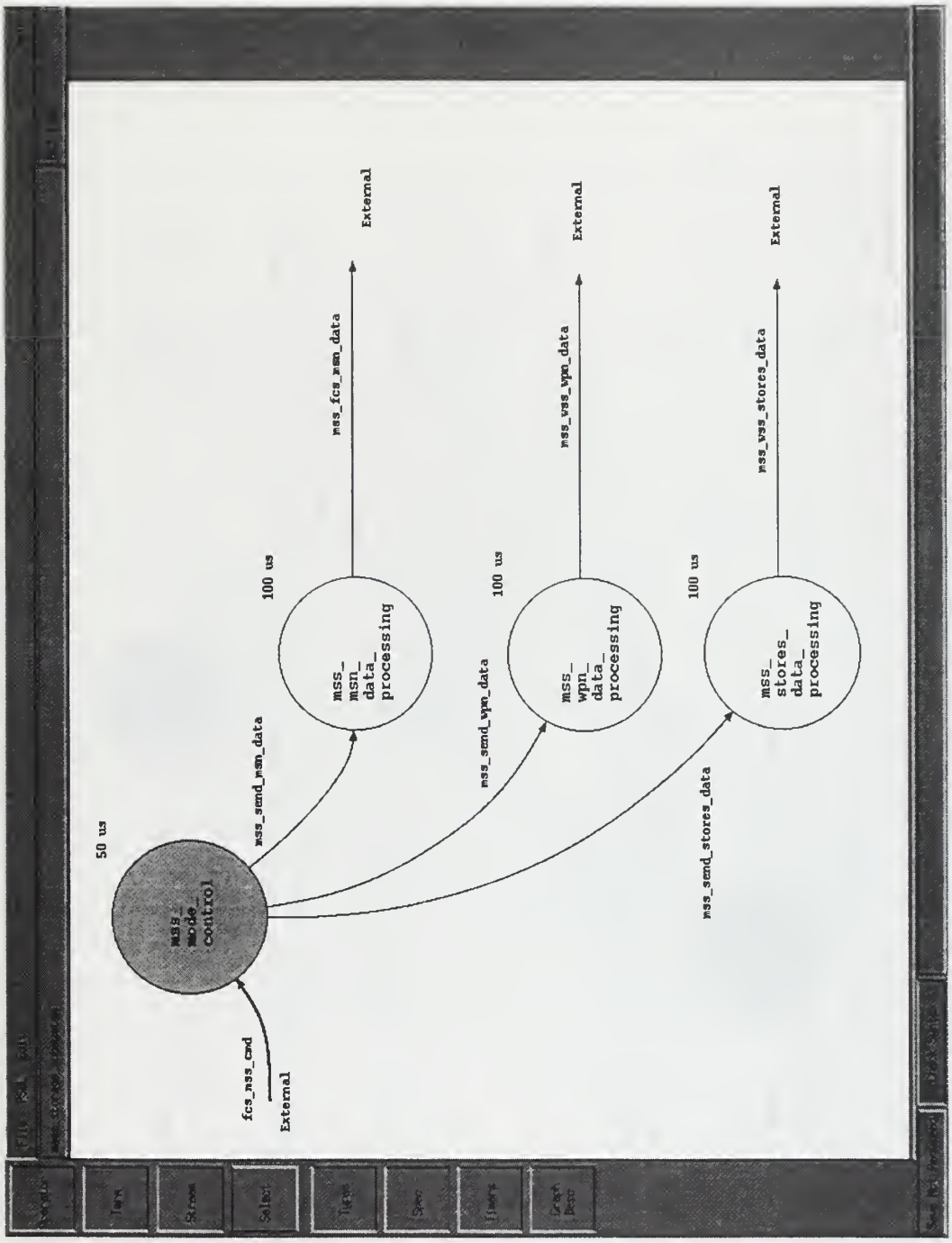


Figure 61. avionics_example MSS Operator

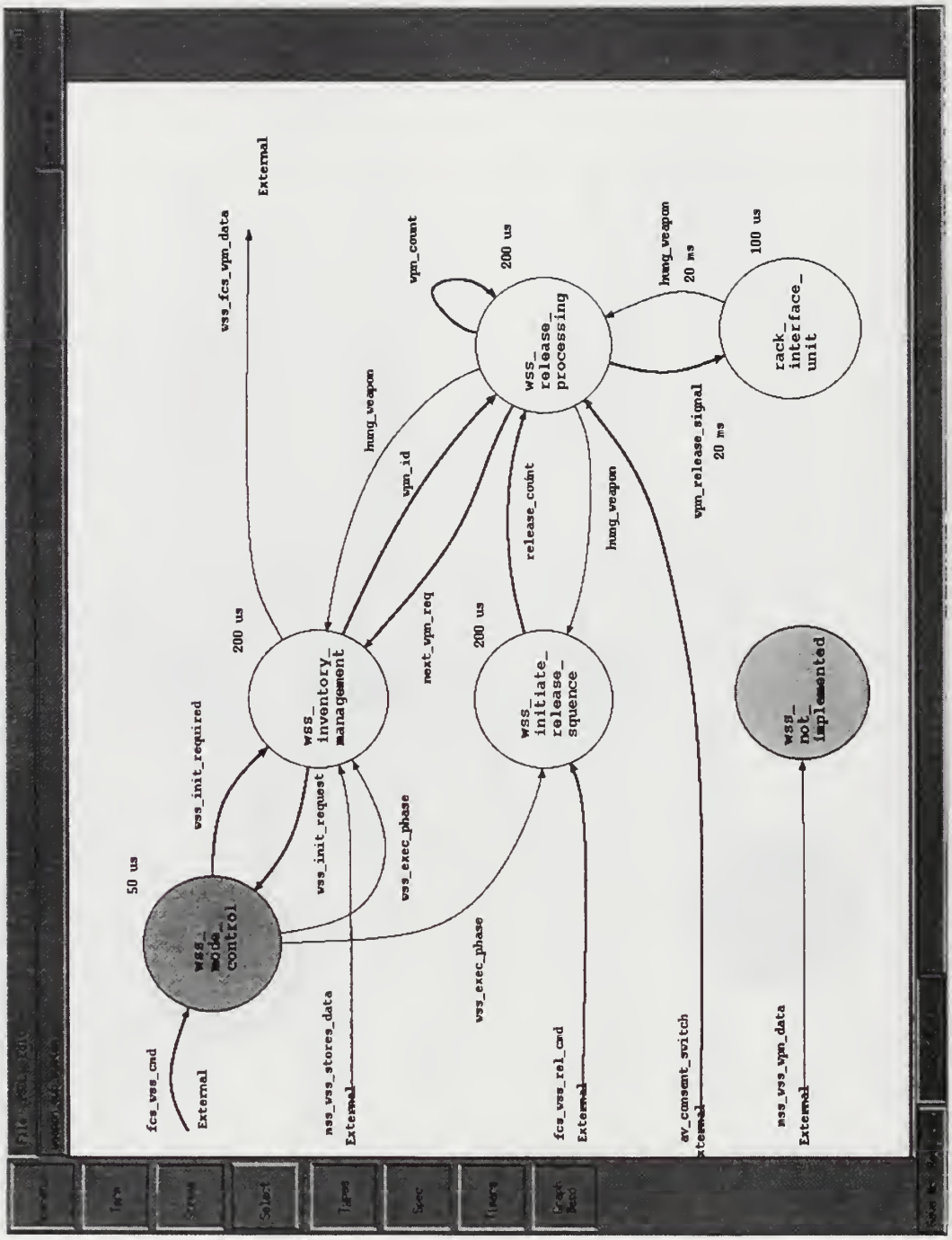


Figure 62. avionics_example WSS Operator

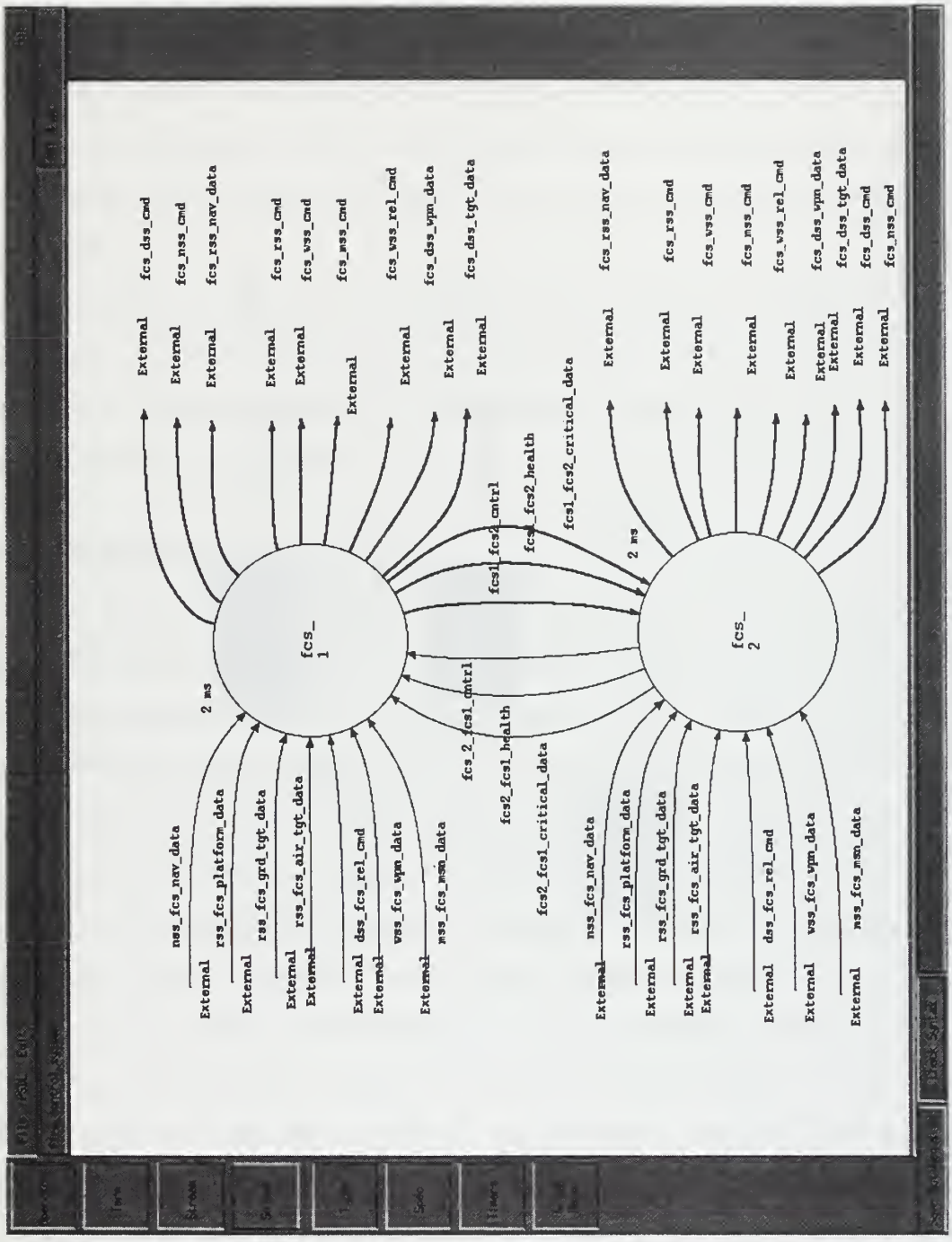


Figure 63. `avionics_example` FCS Operator

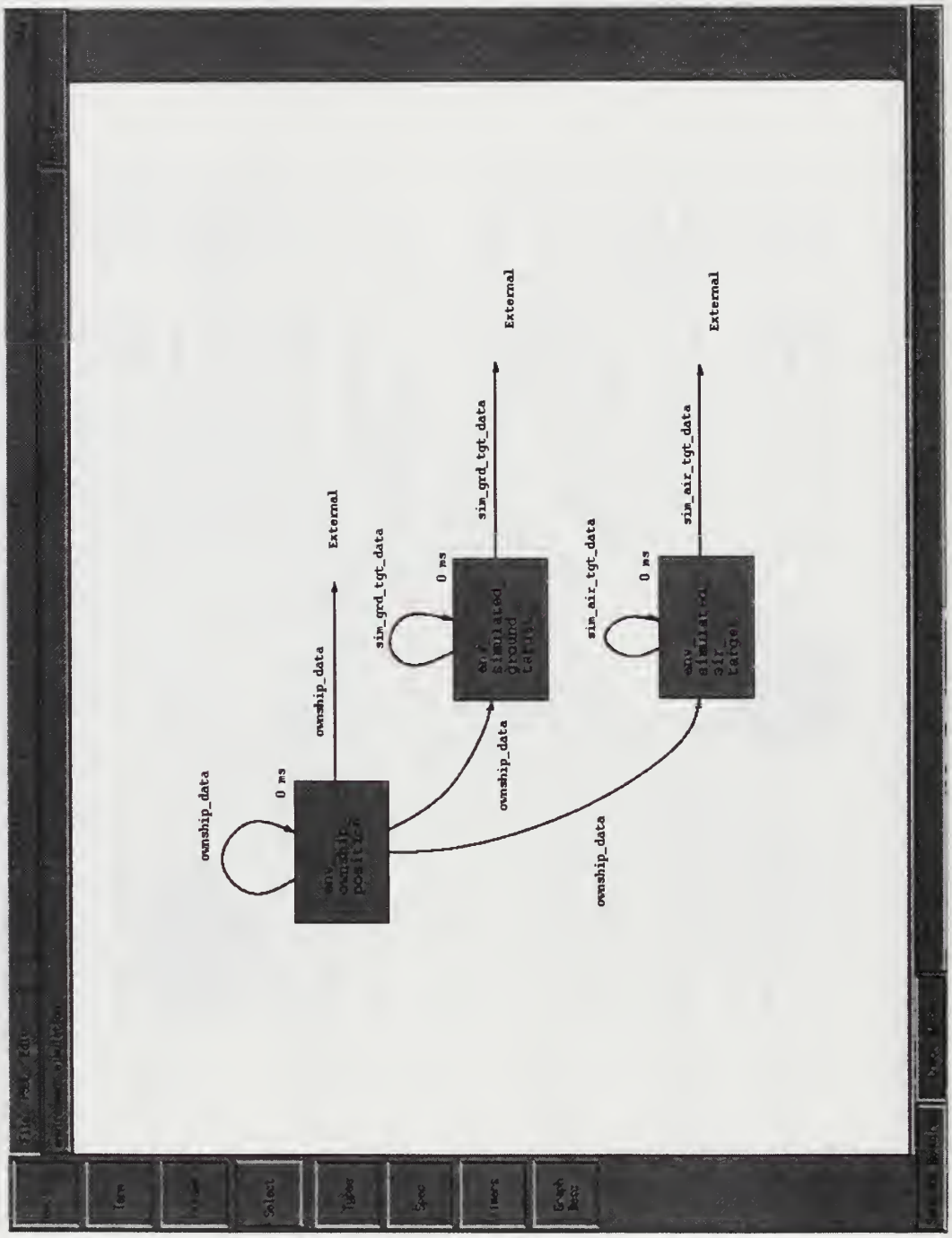


Figure 64. `avionics_example` Environment Operator

each of these ⟨*operator*⟩ ⟨*component*⟩'s are reported as multiple root operators.

After using a text editor to correct these problems, no additional errors were reported by the translator. However, this does not imply that there are no other compatibility issues. Only that no other errors were detected. At this point, the prototype was scheduled. Only one additional error was detected by the scheduler. The scheduler raised a `TIME_INFO_ERROR` exception for having a zero `MET` and a non-zero `PERIOD`.

No additional testing was performed beyond obtaining a valid schedule. However, additional compatibility issues probably still exist but not reported. The lack of operator and stream identification numbers within ⟨*control_constrains*⟩ will most likely cause additional problems.

5. LESSONS LEARNED

The early use of the `avionics_example` prototype in the development of the PSDL Editor was extremely beneficial. Problems were detected with the PSDL Editor that required major rework of the system while there was still time in the schedule to accommodate changes. Specifically, it was found that the PSDL Editor required a large amount of time to perform syntax checks and took an enormous amount of system memory. In addition, the protocol used to exchange information between the background checker and the graph editor failed for large prototypes. Neither of these problems were readily apparent from the use of smaller test cases.

In stepping from a small prototype to a large prototype, it was apparent that the PSDL Editor required too much time to traverse the prototype. It was typical to take 45 seconds to move from one level of the prototype to another. Timing analysis indicated that the largest component of this delay was due to updating the attribute tree maintained by the background checker, which uses Synthesizer Generator routines to maintain the attribute tree as in CAPS Release 1. However, each of these routines also updated the syntax-directed editor window, which resulted in even longer delays.

The syntax-directed editor window is not used in this version of the PSDL Editor, and thus resulted in wasted processor time. System performance was improved through the redesign of the background checker. A new Ada/C background checker, which did not use the Synthesizer Generator code, is now used to maintain the prototype data structure. This redesign resulted in greatly improved performance during the traversal of the prototype and significant improvements in the delays encountered during syntax checks and save operations.

In the process of working with the `avionics_example`, it was found that the syntax-directed editor reported being out of memory (virtual memory swap space). The problem appeared to be one of accumulation. The syntax-directed editor started with a reasonably sized workspace. However, as the user traversed the prototype, the memory requirements grew until the available swap space was no longer able to accommodate the program. This problem is now solved with the new background checker.

Between the background checker process and the graph editor process, two pipes are used to provide bidirectional communication. The protocol used to communicate worked fine for small prototypes. However, when the `avionics_example` prototype was used, this protocol fell apart. The error resulted from the maximum limit on the size of read and write operations on the pipe. Using the larger prototype, this limit was exceeded. A solution was obtained by transmitting the buffer used for inter-process communication using multiple packets of a maximum size reflective of the pipe limitations.

Besides these errors, modifications to the user interface were identified to improve user interaction. Once again, these modifications were identified after repeated use of the editor. They were not readily apparent from the entry of one or two operators.

6. AVIONICS EXAMPLE PSDL CODE

The following is the PSDL code generated by the PSDL Editor for the prototype `avionics_example`.

```
TYPE delta_tgt_format
SPECIFICATION
END
IMPLEMENTATION Ada delta_tgt_format

END

TYPE empty_inventory
SPECIFICATION
END
IMPLEMENTATION Ada empty_inventory

END

TYPE exception_type
SPECIFICATION
END
IMPLEMENTATION Ada exception_type

END

TYPE fcs_cntrl_format
SPECIFICATION
END
IMPLEMENTATION Ada fcs_cntrl_format

END

TYPE fcs_critical_format
SPECIFICATION
END
IMPLEMENTATION Ada fcs_critical_format

END

TYPE fcs_health_format
SPECIFICATION
END
IMPLEMENTATION Ada fcs_health_format

END

TYPE fcs_status_format
SPECIFICATION
END
IMPLEMENTATION Ada fcs_status_format

END

TYPE initial_air_target
SPECIFICATION
END
IMPLEMENTATION Ada initial_air_target

END

TYPE initial_fcs_cntrl
SPECIFICATION
END
IMPLEMENTATION Ada initial_fcs_cntrl

END

TYPE initial_fcs_critical
SPECIFICATION
END
IMPLEMENTATION Ada initial_fcs_critical

END

TYPE initial_fcs_health
SPECIFICATION
END
IMPLEMENTATION Ada initial_fcs_health

END

TYPE initial_fcs_status
SPECIFICATION
END
IMPLEMENTATION Ada initial_fcs_status

END

TYPE initial_grd_target
SPECIFICATION
END
IMPLEMENTATION Ada initial_grd_target

END

TYPE initial_position
SPECIFICATION
END
IMPLEMENTATION Ada initial_position

END

TYPE msn_format
SPECIFICATION
END
IMPLEMENTATION Ada msn_format

END

TYPE nav_format
```

```
SPECIFICATION
END
IMPLEMENTATION Ada nav_format

END

TYPE no_weapon_selected
SPECIFICATION
END
IMPLEMENTATION Ada no_weapon_selected

END

TYPE Rate_1Hz
SPECIFICATION
END
IMPLEMENTATION Ada Rate_1Hz

END

TYPE Rate_25Hz
SPECIFICATION
END
IMPLEMENTATION Ada Rate_25Hz

END

TYPE stores_format
SPECIFICATION
END
IMPLEMENTATION Ada stores_format

END

TYPE wpn_format
SPECIFICATION
END
IMPLEMENTATION Ada wpn_format

END

TYPE wpn_id_format
SPECIFICATION
END
IMPLEMENTATION Ada wpn_id_format

END

TYPE zero_position
SPECIFICATION
END
IMPLEMENTATION Ada zero_position

END

TYPE zero_tgt_position
SPECIFICATION
END
IMPLEMENTATION Ada zero_tgt_position

END

OPERATOR air_vehicle_interface_2584
SPECIFICATION
    INPUT
      av_displays_wpn : wpn_format
    INPUT
      av_displays_tgt : delta_tgt_format
    INPUT
      av_display_fcs : fcs_status_format
    OUTPUT
      av_consent_switch : BOOLEAN
    OUTPUT
      av_rel_switch : BOOLEAN
    MAXIMUM EXECUTION TIME 0 MS
END
IMPLEMENTATION TAE_PLUS air_vehicle_interface_2584

END

OPERATOR avionics_example_2543
SPECIFICATION
    STATES
      fcs_dss_tgt_data : delta_tgt_format
      INITIALLY
        zero_tgt_position
    STATES
      fcs_dss_cmd : fcs_status_format
      INITIALLY
        initial_fcs_status
    STATES
      fcs_nss_cmd : fcs_status_format
      INITIALLY
        initial_fcs_status
    STATES
      fcs_wss_rel_cmd : BOOLEAN
      INITIALLY
        FALSE
    STATES
      av_rel_switch : BOOLEAN
      INITIALLY
        FALSE
    STATES
      av_consent_switch : BOOLEAN
      INITIALLY
        FALSE
```

```
    STATES
      fcs_wss_cmd : fcs_status_format
      INITIALLY
        initial_fcs_status
    STATES
      fcs_mss_cmd : fcs_status_format
      INITIALLY
        initial_fcs_status
    STATES
      fcs_dss_wpn_data : wpn_format
      INITIALLY
        empty_inventory
    STATES
      fcs_rss_nav_data : nav_format
      INITIALLY
        zero_position
    STATES
      fcs_rss_cmd : fcs_status_format
      INITIALLY
        initial_fcs_status
    STATES
      sim_air_tgt_data : nav_format
      INITIALLY
        initial_air_target
    STATES
      sim_grd_tgt_data : nav_format
      INITIALLY
        initial_grd_target
    STATES
      ownship_data : nav_format
      INITIALLY
        initial_position
END
IMPLEMENTATION
  GRAPH
    VERTEX radar_sub_system_2577_2544 : 3 MS
      PROPERTY x = 56
      PROPERTY y = 285
      PROPERTY radius = 63
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 36
      PROPERTY label_y_offset = 86
      PROPERTY met_font = 2
      PROPERTY met_x_offset =  - 34
      PROPERTY met_y_offset = 28
      PROPERTY is_terminator = FALSE

    VERTEX nav_sub_system_2578_2545 : 1 MS
      PROPERTY x = 387
      PROPERTY y = 23
      PROPERTY radius = 64
      PROPERTY color = 63
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 37
      PROPERTY label_y_offset = 89
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 123
      PROPERTY met_y_offset = 13
      PROPERTY is_terminator = FALSE

    VERTEX weapon_sub_system_2579_2546 : 5 MS
      PROPERTY x = 808
      PROPERTY y = 287
      PROPERTY radius = 64
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 34
      PROPERTY label_y_offset = 87
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 107
      PROPERTY met_y_offset = 4
      PROPERTY is_terminator = FALSE

    VERTEX display_sub_system_2580_2547 : 4 MS
      PROPERTY x = 255
      PROPERTY y = 543
      PROPERTY radius = 64
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 28
      PROPERTY label_y_offset = 85
      PROPERTY met_font = 2
      PROPERTY met_x_offset =  - 25
      PROPERTY met_y_offset = 22
      PROPERTY is_terminator = FALSE

    VERTEX mass_storage_subsystem_2581_2548 : 1 MS
      PROPERTY x = 566
      PROPERTY y = 541
      PROPERTY radius = 64
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 26
      PROPERTY label_y_offset = 87
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 110
      PROPERTY met_y_offset = 5
      PROPERTY is_terminator = FALSE

    VERTEX fire_control_system_2582_2549 : 4 MS
      PROPERTY x = 402
      PROPERTY y = 280
      PROPERTY radius = 64
      PROPERTY color = 62
      PROPERTY label_font = 3
```

```
  PROPERTY label_x_offset = 27
  PROPERTY label_y_offset = 84
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 107
  PROPERTY met_y_offset = 6
  PROPERTY is_terminator = FALSE

VERTEX environment_simulation_2583_2550 : 0 MS
  PROPERTY x = 43
  PROPERTY y = 30
  PROPERTY radius = 54
  PROPERTY color = 7
  PROPERTY label_font = 6
  PROPERTY label_x_offset = 26
  PROPERTY label_y_offset = 70
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 130
  PROPERTY met_y_offset =  - 8
  PROPERTY is_terminator = TRUE

VERTEX air_vehicle_interface_2584_2551 : 0 MS
  PROPERTY x = 788
  PROPERTY y = 557
  PROPERTY radius = 54
  PROPERTY color = 7
  PROPERTY label_font = 6
  PROPERTY label_x_offset = 40
  PROPERTY label_y_offset = 73
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 142
  PROPERTY met_y_offset =  - 11
  PROPERTY is_terminator = TRUE

EDGE ownship_data
  environment_simulation_2583_2550 ->
  nav_sub_system_2578_2545
    PROPERTY id = 2554
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 44
    PROPERTY label_y_offset = 25
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "237 57 361 56 "

EDGE ownship_data
  environment_simulation_2583_2550 ->
  radar_sub_system_2577_2544
    PROPERTY id = 2555
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 2
    PROPERTY label_y_offset =  - 49
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "182 169 182 262 "

EDGE sim_grd_tgt_data
  environment_simulation_2583_2550 ->
  radar_sub_system_2577_2544
    PROPERTY id = 2556
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 9
    PROPERTY label_y_offset = 9
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "124 172 125 257 "

EDGE sim_air_tgt_data
  environment_simulation_2583_2550 ->
  radar_sub_system_2577_2544
    PROPERTY id = 2557
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 11
    PROPERTY label_y_offset = 48
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "69 161 68 260 "

EDGE fcs_rss_cmd
  fire_control_system_2582_2549 ->
  radar_sub_system_2577_2544
    PROPERTY id = 2558
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 48
    PROPERTY label_y_offset =  - 7
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "385 268 233 269 "

EDGE fcs_rss_nav_data
  fire_control_system_2582_2549 ->
  radar_sub_system_2577_2544
    PROPERTY id = 2559
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 50
    PROPERTY label_y_offset =  - 7
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "356 306 241 306 "

EDGE rss_fcs_grd_tgt_data
```

```
  radar_sub_system_2577_2544 ->
  fire_control_system_2582_2549
    PROPERTY id = 2560
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 69
    PROPERTY label_y_offset = 17
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "212 365 358 364 "

EDGE rss_fcs_air_tgt_data
  radar_sub_system_2577_2544 ->
  fire_control_system_2582_2549
    PROPERTY id = 2561
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 60
    PROPERTY label_y_offset = 14
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "216 399 345 398 "

EDGE rss_fcs_platform_data
  radar_sub_system_2577_2544 ->
  fire_control_system_2582_2549
    PROPERTY id = 2562
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 89
    PROPERTY label_y_offset = 18
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "217 437 362 436 "

EDGE dss_fcs_rel_cmd
  display_sub_system_2580_2547 ->
  fire_control_system_2582_2549
    PROPERTY id = 2563
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 140
    PROPERTY label_y_offset = 47
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "340 515 391 430 "

EDGE fcs_dss_wpn_data
  fire_control_system_2582_2549 ->
  display_sub_system_2580_2547
    PROPERTY id = 2564
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 22
    PROPERTY label_y_offset = 52
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "465 446 409 547 "

EDGE fcs_mss_cmd
  fire_control_system_2582_2549 ->
  mass_storage_subsystem_2581_2548
    PROPERTY id = 2565
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "487 449 545 534 "

EDGE fcs_wss_cmd
  fire_control_system_2582_2549 ->
  weapon_sub_system_2579_2546
    PROPERTY id = 2567
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 43
    PROPERTY label_y_offset =  - 5
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "561 298 752 299 "

EDGE wss_fcs_wpn_data
  weapon_sub_system_2579_2546 ->
  fire_control_system_2582_2549
    PROPERTY id = 2568
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 64
    PROPERTY label_y_offset = 19
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "760 385 596 382 "

EDGE mss_wss_stores_data
  mass_storage_subsystem_2581_2548 ->
  weapon_sub_system_2579_2546
    PROPERTY id = 2569
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 10
    PROPERTY label_y_offset = 11
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "660 516 744 412 "
```

```
EDGE av_consent_switch
  air_vehicle_interface_2584_2551 ->
  weapon_sub_system_2679_2546
    PROPERTY id = 2571
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 138
    PROPERTY label_y_offset = 60
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "925 623 923 444 "

EDGE av_rel_switch
  air_vehicle_interface_2584_2551 ->
  display_sub_system_2580_2547
    PROPERTY id = 2572
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 37
    PROPERTY label_y_offset = 17
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "825 723 744 743 450 744 391 720 "

EDGE av_displays_wpn
  display_sub_system_2580_2647 ->
  air_vehicle_interface_2684_2551
    PROPERTY id = 2573
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 69
    PROPERTY label_y_offset = 15
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "398 680 452 698 756 695 789 679 "

EDGE nss_fcs_nav_data
  nav_sub_system_2578_2545 ->
  fire_control_system_2582_2549
    PROPERTY id = 2574
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 12
    PROPERTY label_y_offset = 1
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "495 180 496 247 "

EDGE fcs_wss_rel_cmd
  fire_control_system_2582_2549 ->
  weapon_sub_system_2579_2546
    PROPERTY id = 2612
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 33
    PROPERTY label_y_offset =  - 6
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "653 327 714 324 "

EDGE fcs_nss_cmd
  fire_control_system_2582_2549 ->
  nav_sub_system_2578_2545
    PROPERTY id = 2746
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 86
    PROPERTY label_y_offset =  - 49
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "409 244 408 175 "

EDGE fcs_dss_cmd
  fire_control_system_2582_2549 ->
  display_sub_system_2580_2647
    PROPERTY id = 2747
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 1
    PROPERTY label_y_offset = 21
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "430 437 378 534 "

EDGE mss_fcs_msn_data
  mass_storage_subsystem_2581_2548 ->
  fire_control_system_2582_2549
    PROPERTY id = 2798
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "618 496 546 399 "

EDGE fcs_dss_tgt_data
  fire_control_system_2582_2549 ->
  display_sub_system_2580_2547
    PROPERTY id = 2826
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 55
    PROPERTY label_y_offset = 66
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
```

```
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "480 461 457 525 421 563 "

  EDGE av_displays_tgt
    display_sub_system_2580_2547 ->
    air_vehicle_interface_2584_2551
      PROPERTY id = 2834
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 79
      PROPERTY label_y_offset = 13
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "410 663 466 676 734 676 "

  EDGE av_display_fcs
    display_sub_system_2580_2547 ->
    air_vehicle_interface_2584_2551
      PROPERTY id = 2764
      PROPERTY label_font = 2
      PROPERTY label_x_offset =  - 46
      PROPERTY label_y_offset = 14
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "394 696 428 706 472 720 715 718 782 705 821 682 "

  EDGE mss_wss_wpn_data
    mass_storage_subsystem_2581_2548 ->
    weapon_sub_system_2579_2546
      PROPERTY id = 2787
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 5
      PROPERTY label_y_offset = 14
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "719 557 836 451 "

DATA STREAM
  mss_wss_wpn_data : BOOLEAN,
  av_display_fcs : fcs_status_format,
  av_displays_tgt : delta_tgt_format,
  mss_fcs_msn_data : msn_format,
  nss_fcs_nav_data : nav_format,
  av_displays_wpn : wpn_format,
  mss_wss_stores_data : stores_format,
  wss_fcs_wpn_data : wpn_format,
  dss_fcs_rel_cmd : BOOLEAN,
  rss_fcs_platform_data : nav_format,
  rss_fcs_air_tgt_data : delta_tgt_format,
  rss_fcs_grd_tgt_data : delta_tgt_format
CONTROL CONSTRAINTS
      OPERATOR radar_sub_system_2577_2544

      OPERATOR nav_sub_system_2578_2545

      OPERATOR weapon_sub_system_2579_2546

      OPERATOR display_sub_system_2580_2547

      OPERATOR mass_storage_subsystem_2581_2548

      OPERATOR fire_control_system_2582_2549

      OPERATOR environment_simulation_2583_2550
        PERIOD 20 MS

      OPERATOR air_vehicle_interface_2584_2551
END

OPERATOR display_sub_system_2580
SPECIFICATION
    INPUT
      fcs_dss_wpn_data : wpn_format
    INPUT
      av_rel_switch : BOOLEAN
    INPUT
      fcs_dss_cmd : fcs_status_format
    INPUT
      fcs_dss_tgt_data : delta_tgt_format
    OUTPUT
      dss_fcs_rel_cmd : BOOLEAN
    OUTPUT
      av_displays_wpn : wpn_format
    OUTPUT
      av_displays_tgt : delta_tgt_format
    OUTPUT
      av_display_fcs : fcs_status_format
    STATES
      dss_mode_phase : Rate_1Hz
      INITIALLY
        0
    MAXIMUM EXECUTION TIME 4 MS
END
IMPLEMENTATION
  GRAPH
    VERTEX dss_weapon_display_processing_2677_2667 : 100 MICROSEC
      PROPERTY x = 383
      PROPERTY y = 406
      PROPERTY radius = 75
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 32
      PROPERTY label_y_offset = 105
```

```
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 129
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX dss_rel_switch_processing_2678_2668 : 100 MICROSEC
  PROPERTY x = 382
  PROPERTY y = 222
  PROPERTY radius = 74
  PROPERTY color = 62
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 31
  PROPERTY label_y_offset = 104
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 127
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX dss_mode_control_2679_2673 : 50 MICROSEC
  PROPERTY x = 101
  PROPERTY y = 73
  PROPERTY radius = 75
  PROPERTY color = 28
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 47
  PROPERTY label_y_offset = 98
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 133
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX dss_target_display_processing_2833_2827 : 100 MICROSEC
  PROPERTY x = 386
  PROPERTY y = 592
  PROPERTY radius = 77
  PROPERTY color = 62
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 34
  PROPERTY label_y_offset = 106
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 133
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

EDGE av_rel_switch
  EXTERNAL ->
  dss_rel_switch_processing_2678_2668
    PROPERTY id = 2669
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 308
    PROPERTY label_y_offset =  - 13
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "89 357 316 357 363 343 "

EDGE dss_fcs_rel_cmd
  dss_rel_switch_processing_2678_2668 ->
  EXTERNAL
    PROPERTY id = 2670
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 80
    PROPERTY label_y_offset =  - 10
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "562 293 768 294 "

EDGE fcs_dss_wpn_data
  EXTERNAL ->
  dss_weapon_display_processing_2677_2667
    PROPERTY id = 2671
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 273
    PROPERTY label_y_offset =  - 19
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "101 526 241 526 354 527 "

EDGE av_displays_wpn
  dss_weapon_display_processing_2677_2667 ->
  EXTERNAL
    PROPERTY id = 2672
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 65
    PROPERTY label_y_offset =  - 11
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "589 480 776 481 "

EDGE dss_exec_phase_all
  dss_mode_control_2679_2673 ->
  dss_rel_switch_processing_2678_2668
    PROPERTY id = 2674
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 62
    PROPERTY label_y_offset =  - 53
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "262 290 348 299 "

EDGE dss_exec_phase_1Hz
  dss_mode_control_2679_2673 ->
```

```
    dss_weapon_display_processing_2677_2667
      PROPERTY id = 2675
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 17
      PROPERTY label_y_offset = 11
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "227 330 317 465 "

  EDGE dss_mode_phase
    dss_mode_control_2679_2673 ->
    dss_mode_control_2679_2673
      PROPERTY id = 2676
      PROPERTY label_font = 2
      PROPERTY label_x_offset =  - 48
      PROPERTY label_y_offset =  - 8
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "64 91 69 45 105 52 "

  EDGE av_displays_tgt
    dss_target_display_processing_2833_2827 ->
    EXTERNAL
      PROPERTY id = 2828
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 48
      PROPERTY label_y_offset =  - 14
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "575 667 765 667 "

  EDGE dss_exec_phase_1Hz
    dss_mode_control_2679_2673 ->
    dss_target_display_processing_2833_2827
      PROPERTY id = 2829
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 60
      PROPERTY label_y_offset = 40
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "179 364 202 576 305 672 "

  EDGE fcs_dss_tgt_data
    EXTERNAL ->
    dss_target_display_processing_2833_2827
      PROPERTY id = 2831
      PROPERTY label_font = 2
      PROPERTY label_x_offset =  - 303
      PROPERTY label_y_offset =  - 15
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "94 731 298 728 370 723 "

    EDGE fcs_dss_cmd
      EXTERNAL ->
      dss_mode_control_2679_2673
        PROPERTY id = 2832
        PROPERTY label_font = 2
        PROPERTY label_x_offset =  - 85
        PROPERTY label_y_offset =  - 10
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 0
        PROPERTY spline = "42 241 85 240 126 229 "

    EDGE av_displays_fcs
      dss_mode_control_2679_2673 ->
      EXTERNAL
        PROPERTY id = 2763
        PROPERTY label_font = 2
        PROPERTY label_x_offset = 133
        PROPERTY label_y_offset =  - 15
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 15
        PROPERTY spline = "508 148 763 149 "

  DATA STREAM
    av_displays_fcs : fcs_status_format,
    dss_exec_phase_1Hz : BOOLEAN,
    dss_exec_phase_all : BOOLEAN
  CONTROL CONSTRAINTS

      OPERATOR dss_weapon_display_processing_2677_2667
        MINIMUM CALLING PERIOD 20 MS

      OPERATOR dss_rel_switch_processing_2678_2668
        MINIMUM CALLING PERIOD 20 MS

      OPERATOR dss_mode_control_2679_2673
        TRIGGERED BY ALL
          fcs_dss_cmd
        MINIMUM CALLING PERIOD 20 MS
        OUTPUT
          dss_exec_phase_all
          IF TRUE

        OUTPUT
          dss_exec_phase_1Hz
          IF dss_mode_phase = 49
```

```
      OPERATOR dss_target_display_processing_2833_2827
        MINIMUM CALLING PERIOD 20 MS
END

OPERATOR dss_mode_control_2679
SPECIFICATION
    INPUT
      dss_mode_phase : Rate_1Hz
    INPUT
      fcs_dss_cmd : fcs_status_format
    OUTPUT
      dss_exec_phase_all : BOOLEAN
    OUTPUT
      dss_exec_phase_1Hz : BOOLEAN
    OUTPUT
      dss_mode_phase : Rate_1Hz
    OUTPUT
      av_displays_fcs : fcs_status_format
    MAXIMUM EXECUTION TIME 50 MICROSEC
END
IMPLEMENTATION ADA dss_mode_control_2679

END

OPERATOR dss_rel_switch_processing_2678
SPECIFICATION
    INPUT
      av_rel_switch : BOOLEAN
    INPUT
      dss_exec_phase_all : BOOLEAN
    OUTPUT
      dss_fcs_rel_cmd : BOOLEAN
    MAXIMUM EXECUTION TIME 100 MICROSEC
END
IMPLEMENTATION ADA dss_rel_switch_processing_2678

END

OPERATOR dss_target_display_processing_2833
SPECIFICATION
    INPUT
      dss_exec_phase_1Hz : BOOLEAN
    INPUT
      fcs_dss_tgt_data : delta_tgt_format
    OUTPUT
      av_displays_tgt : delta_tgt_format
    MAXIMUM EXECUTION TIME 100 MICROSEC
END
IMPLEMENTATION ADA dss_target_display_processing_2833

END

OPERATOR dss_weapon_display_processing_2677
SPECIFICATION
    INPUT
      fcs_dss_wpn_data : wpn_format
    INPUT
      dss_exec_phase_1Hz : BOOLEAN
    OUTPUT
      av_displays_wpn : wpn_format
    MAXIMUM EXECUTION TIME 100 MICROSEC
END
IMPLEMENTATION ADA dss_weapon_display_processing_2677

END

OPERATOR env_ownship_position_2625
SPECIFICATION
    INPUT
      ownship_data : nav_format
    OUTPUT
      ownship_data : nav_format
    MAXIMUM EXECUTION TIME 0 MS
END
IMPLEMENTATION ADA env_ownship_position_2625

END

OPERATOR env_simulated_air_target_2627
SPECIFICATION
    INPUT
      ownship_data : nav_format
    INPUT
      sim_air_tgt_data : nav_format
    OUTPUT
      sim_air_tgt_data : nav_format
    MAXIMUM EXECUTION TIME 0 MS
END
IMPLEMENTATION ADA env_simulated_air_target_2627

END

OPERATOR env_simulated_ground_target_2626
SPECIFICATION
    INPUT
      ownship_data : nav_format
    INPUT
      sim_grd_tgt_data : nav_format
    OUTPUT
      sim_grd_tgt_data : nav_format
    MAXIMUM EXECUTION TIME 0 MS
    KEYWORDS
      Environment, state_vector
    DESCRIPTION
      {**** Flat earth air vehicle simulation:  This operator will provide
       the avionics_example prototype with a dynamic air vehicle.  While
```

```
        dynamic motion will be simulated, the course taken by the air
        vehicle will be static.  This model provides for a constant
        flight heading, velocity, and altitude.  No accelerations,
        translational or rotational, will be available to the flight
        path.
        **** The coordinate system used operator will be based on a flat
        earth.  X is positive East, Y is positive North, Z is positive Up.
        Heading will be measured from the X axis, positive Counter-Clockwise. }
    AXIOMS
      {Z > 0
       Heading = constant
       Roll = 0
       Pitch = 0
       Yaw   = 0
       velocity = constant }
END
IMPLEMENTATION ADA env_simulated_ground_target_2626

END

OPERATOR environment_simulation_2583
SPECIFICATION
    OUTPUT
      ownship_data : nav_format
    OUTPUT
      sim_grd_tgt_data : nav_format
    OUTPUT
      sim_air_tgt_data : nav_format
    MAXIMUM EXECUTION TIME 0 MS
    KEYWORDS
      simulation, earth_model, ground_target_model, air_target_model,
      flat_earth, periodic
    DESCRIPTION
      {This operator provides the environment simulation in order to
       operate the avionics prototype.  Since the operations performed
       by this operator do not correspond to processing performed, but
       instead are a simulation of a physical process, the execution
       time of the operator is not counted.
       A flat earth model is used throughout this simulation.
       X -->  East,
       Y -->  North,
       Z -->  Up.
       This model is only applicable to the Northern Hemisphere.
       The ground is at Z = 0. }
    AXIOMS
      {Ownship(Z) >= 0
       AirTarget(Z) >= 0
       GroundTarget(Z) = 0 }
END
IMPLEMENTATION
  GRAPH
    VERTEX env_ownship_position_2625_2613 : 0 MS
      PROPERTY x = 214
      PROPERTY y = 202
      PROPERTY radius = 44
      PROPERTY color = 7
      PROPERTY label_font = 6
      PROPERTY label_x_offset = 27
      PROPERTY label_y_offset = 63
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 116
      PROPERTY met_y_offset =  - 8
      PROPERTY is_terminator = TRUE

    VERTEX env_simulated_ground_target_2626_2614 : 0 MS
      PROPERTY x = 421
      PROPERTY y = 351
      PROPERTY radius = 44
      PROPERTY color = 7
      PROPERTY label_font = 6
      PROPERTY label_x_offset = 25
      PROPERTY label_y_offset = 73
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 110
      PROPERTY met_y_offset =  - 6
      PROPERTY is_terminator = TRUE

    VERTEX env_simulated_air_target_2627_2615 : 0 MS
      PROPERTY x = 423
      PROPERTY y = 542
      PROPERTY radius = 44
      PROPERTY color = 7
      PROPERTY label_font = 6
      PROPERTY label_x_offset = 21
      PROPERTY label_y_offset = 73
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 109
      PROPERTY met_y_offset =  - 8
      PROPERTY is_terminator = TRUE

    EDGE ownship_data
      env_ownship_position_2625_2613 ->
      env_ownship_position_2625_2613
        PROPERTY id = 2617
        PROPERTY label_font = 2
        PROPERTY label_x_offset = 0
        PROPERTY label_y_offset = 0
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 15
        PROPERTY spline = "232 164 271 124 319 160 "

    EDGE ownship_data
      env_ownship_position_2625_2613 ->
      env_simulated_ground_target_2626_2614
        PROPERTY id = 2618
```

```
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 45
    PROPERTY label_y_offset = 36
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "330 347 396 383 "

EDGE ownship_data
  env_ownship_position_2625_2613 ->
  env_simulated_air_target_2627_2615
    PROPERTY id = 2619
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 92
    PROPERTY label_y_offset = 10
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "282 366 321 481 385 575 "

EDGE sim_air_tgt_data
  env_simulated_air_target_2627_2615 ->
  EXTERNAL
    PROPERTY id = 2620
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 21
    PROPERTY label_y_offset =  - 8
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "563 581 734 582 "

EDGE sim_grd_tgt_data
  env_simulated_ground_target_2626_2614 ->
  EXTERNAL
    PROPERTY id = 2621
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 29
    PROPERTY label_y_offset =  - 9
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "560 391 737 392 "

EDGE ownship_data
  env_ownship_position_2625_2613 ->
  EXTERNAL
    PROPERTY id = 2622
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 33
    PROPERTY label_y_offset =  - 11
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "354 240 532 240 "

  EDGE sim_air_tgt_data
    env_simulated_air_target_2627_2615 ->
    env_simulated_air_target_2627_2615
      PROPERTY id = 2623
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 0
      PROPERTY label_y_offset = 0
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "458 517 478 483 504 509 "

  EDGE sim_grd_tgt_data
    env_simulated_ground_target_2626_2614 ->
    env_simulated_ground_target_2626_2614
      PROPERTY id = 2624
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 0
      PROPERTY label_y_offset = 0
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "448 313 469 286 509 302 "
CONTROL CONSTRAINTS

  OPERATOR env_ownship_position_2625_2613
    PERIOD 20 MS
    FINISH WITHIN 200 MICROSEC

  OPERATOR env_simulated_ground_target_2626_2614
    TRIGGERED BY ALL
      ownship_data
      IF TRUE
    PERIOD 20 MS
    FINISH WITHIN 100 MICROSEC
    MINIMUM CALLING PERIOD 20 MS

  OPERATOR env_simulated_air_target_2627_2615
    TRIGGERED BY ALL
      ownship_data
      IF FALSE
    MINIMUM CALLING PERIOD 20 MS
    MAXIMUM RESPONSE TIME 100 MICROSEC
END

OPERATOR fcs_1_2666
SPECIFICATION
  INPUT
    rss_fcs_air_tgt_data : delta_tgt_format
  INPUT
```

```
      rss_fcs_grd_tgt_data : delta_tgt_format
    INPUT
      rss_fcs_platform_data : nav_format
    INPUT
      dss_fcs_rel_cmd : BOOLEAN
    INPUT
      wss_fcs_wpn_data : wpn_format
    INPUT
      mss_fcs_msn_data : msn_format
    INPUT
      nss_fcs_nav_data : nav_format
    INPUT
      fcs_2_fcs1_cntrl : fcs_cntrl_format
    INPUT
      fcs2_fcs1_health : fcs_health_format
    INPUT
      fcs2_fcs1_critical_data : fcs_critical_format
    OUTPUT
      fcs_wss_cmd : fcs_status_format
    OUTPUT
      fcs_rss_cmd : fcs_status_format
    OUTPUT
      fcs_rss_nav_data : nav_format
    OUTPUT
      fcs_mss_cmd : fcs_status_format
    OUTPUT
      fcs_dss_wpn_data : wpn_format
    OUTPUT
      fcs1_fcs2_cntrl : fcs_cntrl_format
    OUTPUT
      fcs1_fcs2_health : fcs_health_format
    OUTPUT
      fcs1_fcs2_critical_data : fcs_critical_format
    OUTPUT
      fcs_dss_tgt_data : delta_tgt_format
    OUTPUT
      fcs_nss_cmd : fcs_status_format
    OUTPUT
      fcs_dss_cmd : fcs_status_format
    OUTPUT
      fcs_wss_rel_cmd : BOOLEAN
    MAXIMUM EXECUTION TIME 2 MS
END
IMPLEMENTATION ADA fcs_1_2666

END

OPERATOR fcs_2_2665
SPECIFICATION
    INPUT
      rss_fcs_air_tgt_data : delta_tgt_format
    INPUT
      rss_fcs_grd_tgt_data : delta_tgt_format
    INPUT
      rss_fcs_platform_data : nav_format
    INPUT
      nss_fcs_nav_data : nav_format
    INPUT
      dss_fcs_rel_cmd : BOOLEAN
    INPUT
      wss_fcs_wpn_data : wpn_format
    INPUT
      mss_fcs_msn_data : msn_format
    INPUT
      fcs1_fcs2_cntrl : fcs_cntrl_format
    INPUT
      fcs1_fcs2_health : fcs_health_format
    INPUT
      fcs1_fcs2_critical_data : fcs_critical_format
    OUTPUT
      fcs_wss_cmd : fcs_status_format
    OUTPUT
      fcs_rss_cmd : fcs_status_format
    OUTPUT
      fcs_rss_nav_data : nav_format
    OUTPUT
      fcs_mss_cmd : fcs_status_format
    OUTPUT
      fcs_dss_wpn_data : wpn_format
    OUTPUT
      fcs_2_fcs1_cntrl : fcs_cntrl_format
    OUTPUT
      fcs2_fcs1_health : fcs_health_format
    OUTPUT
      fcs2_fcs1_critical_data : fcs_critical_format
    OUTPUT
      fcs_dss_tgt_data : delta_tgt_format
    OUTPUT
      fcs_dss_cmd : fcs_status_format
    OUTPUT
      fcs_nss_cmd : fcs_status_format
    OUTPUT
      fcs_wss_rel_cmd : BOOLEAN
    MAXIMUM EXECUTION TIME 2 MS
END
IMPLEMENTATION ADA fcs_2_2665

END

OPERATOR fire_control_system_2582
SPECIFICATION
    INPUT
      rss_fcs_grd_tgt_data : delta_tgt_format
    INPUT
      rss_fcs_air_tgt_data : delta_tgt_format
    INPUT
```

```
      rss_fcs_platform_data : nav_format
    INPUT
      dss_fcs_rel_cmd : BOOLEAN
    INPUT
      wss_fcs_wpn_data : wpn_format
    INPUT
      nss_fcs_nav_data : nav_format
    INPUT
      mss_fcs_msn_data : msn_format
    OUTPUT
      fcs_rss_cmd : fcs_status_format
    OUTPUT
      fcs_rss_nav_data : nav_format
    OUTPUT
      fcs_dss_wpn_data : wpn_format
    OUTPUT
      fcs_mss_cmd : fcs_status_format
    OUTPUT
      fcs_wss_cmd : fcs_status_format
    OUTPUT
      fcs_wss_rel_cmd : BOOLEAN
    OUTPUT
      fcs_nss_cmd : fcs_status_format
    OUTPUT
      fcs_dss_cmd : fcs_status_format
    OUTPUT
      fcs_dss_tgt_data : delta_tgt_format
    STATES
      fcs1_fcs2_critical_data : fcs_critical_format
      INITIALLY
        initial_fcs_critical
    STATES
      fcs1_fcs2_health : fcs_health_format
      INITIALLY
        initial_fcs_health
    STATES
      fcs1_fcs2_cntrl : fcs_cntrl_format
      INITIALLY
        initial_fcs_cntrl
    MAXIMUM EXECUTION TIME 4 MS
END
IMPLEMENTATION
  GRAPH
    VERTEX fcs_2_2665_2628 : 2 MS
      PROPERTY x = 392
      PROPERTY y = 522
      PROPERTY radius = 92
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 76
      PROPERTY label_y_offset = 112
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 160
      PROPERTY met_y_offset = 0
      PROPERTY is_terminator = FALSE

    VERTEX fcs_1_2666_2629 : 2 MS
      PROPERTY x = 387
      PROPERTY y = 131
      PROPERTY radius = 89
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 72
      PROPERTY label_y_offset = 105
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 22
      PROPERTY met_y_offset = - 2
      PROPERTY is_terminator = FALSE

    EDGE rss_fcs_air_tgt_data
      EXTERNAL ->
      fcs_1_2666_2629
        PROPERTY id = 2630
        PROPERTY label_font = 2
        PROPERTY label_x_offset = - 88
        PROPERTY label_y_offset = - 7
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 0
        PROPERTY spline = "167 220 254 220 342 218 "

    EDGE rss_fcs_grd_tgt_data
      EXTERNAL ->
      fcs_1_2666_2629
        PROPERTY id = 2631
        PROPERTY label_font = 2
        PROPERTY label_x_offset = - 106
        PROPERTY label_y_offset = - 9
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 0
        PROPERTY spline = "164 188 252 187 355 187 "

    EDGE rss_fcs_platform_data
      EXTERNAL ->
      fcs_1_2666_2629
        PROPERTY id = 2632
        PROPERTY label_font = 2
        PROPERTY label_x_offset = - 142
        PROPERTY label_y_offset = - 11
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 0
        PROPERTY spline = "160 148 294 147 373 155 "

    EDGE dss_fcs_rel_cmd
```

```
  EXTERNAL ->
  fcs_1_2666_2629
    PROPERTY id = 2633
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 115
    PROPERTY label_y_offset = 22
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "160 250 270 249 352 245 "

EDGE wss_fcs_wpn_data
  EXTERNAL ->
  fcs_1_2666_2629
    PROPERTY id = 2634
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 111
    PROPERTY label_y_offset = 24
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "159 283 258 283 355 276 "

EDGE mss_fcs_msn_data
  EXTERNAL ->
  fcs_1_2666_2629
    PROPERTY id = 2635
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 115
    PROPERTY label_y_offset = 27
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "159 326 267 326 354 309 "

EDGE nss_fcs_nav_data
  EXTERNAL ->
  fcs_1_2666_2629
    PROPERTY id = 2636
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 137
    PROPERTY label_y_offset =  - 16
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "155 110 289 109 370 124 "

EDGE rss_fcs_air_tgt_data
  EXTERNAL ->
  fcs_2_2665_2628
    PROPERTY id = 2637
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 94
    PROPERTY label_y_offset =  - 9
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "160 585 253 587 342 586 "

EDGE rss_fcs_grd_tgt_data
  EXTERNAL ->
  fcs_2_2665_2628
    PROPERTY id = 2638
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 132
    PROPERTY label_y_offset =  - 8
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "157 555 282 554 374 556 "

EDGE rss_fcs_platform_data
  EXTERNAL ->
  fcs_2_2665_2628
    PROPERTY id = 2639
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 136
    PROPERTY label_y_offset =  - 8
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "157 520 282 519 370 526 "

EDGE nss_fcs_nav_data
  EXTERNAL ->
  fcs_2_2665_2628
    PROPERTY id = 2640
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 147
    PROPERTY label_y_offset =  - 16
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "154 488 309 489 386 506 "

EDGE dss_fcs_rel_cmd
  EXTERNAL ->
  fcs_2_2665_2628
    PROPERTY id = 2641
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 116
    PROPERTY label_y_offset = 20
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "151 629 266 628 357 624 "
```

```
EDGE wss_fcs_wpn_data
  EXTERNAL ->
  fcs_2_2665_2628
    PROPERTY id = 2642
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 98
    PROPERTY label_y_offset = 22
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "152 667 265 666 339 660 "

EDGE mss_fcs_msn_data
  EXTERNAL ->
  fcs_2_2665_2628
    PROPERTY id = 2643
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 127
    PROPERTY label_y_offset = 29
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "148 709 281 708 391 688 "

EDGE fcs_wss_cmd
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2644
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 259
    PROPERTY label_y_offset = 9
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "550 212 683 212 "

EDGE fcs_rss_cmd
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2645
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 237
    PROPERTY label_y_offset = 8
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "573 186 681 185 "

EDGE fcs_rss_nav_data
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2647
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 246
    PROPERTY label_y_offset = 4
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "560 131 681 131 "

EDGE fcs_mss_cmd
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2648
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 230
    PROPERTY label_y_offset = 18
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "579 236 685 246 "

EDGE fcs_dss_wpn_data
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2650
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 167
    PROPERTY label_y_offset = 2
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "638 333 689 336 "

EDGE fcs_wss_cmd
  fcs_2_2665_2628 ->
  EXTERNAL
    PROPERTY id = 2651
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 210
    PROPERTY label_y_offset = 13
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "613 579 693 579 "

EDGE fcs_rss_cmd
  fcs_2_2665_2628 ->
  EXTERNAL
    PROPERTY id = 2652
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 197
    PROPERTY label_y_offset = 2
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
```

```
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "630 554 697 549 "

EDGE fcs_rss_nav_data
  fcs_2_2665_2628 ->
  EXTERNAL
    PROPERTY id = 2655
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 204
    PROPERTY label_y_offset =  - 6
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "615 503 700 497 "

EDGE fcs_mss_cmd
  fcs_2_2665_2628 ->
  EXTERNAL
    PROPERTY id = 2656
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 179
    PROPERTY label_y_offset = 15
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "644 613 693 615 "

EDGE fcs_dss_wpn_data
  fcs_2_2665_2628 ->
  EXTERNAL
    PROPERTY id = 2658
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 197
    PROPERTY label_y_offset = 17
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "622 676 687 679 "

EDGE fcs_2_fcs1_cntrl
  fcs_2_2665_2628 ->
  fcs_1_2666_2629
    PROPERTY id = 2659
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 115
    PROPERTY label_y_offset =  - 36
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "462 462 459 349 "

EDGE fcs2_fcs1_health
  fcs_2_2665_2628 ->
  fcs_1_2666_2629
    PROPERTY id = 2660
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 119
    PROPERTY label_y_offset = 4
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "431 476 423 400 431 338 "

EDGE fcs2_fcs1_critical_data
  fcs_2_2665_2628 ->
  fcs_1_2666_2629
    PROPERTY id = 2661
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 161
    PROPERTY label_y_offset = 41
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "396 479 381 397 403 327 "

EDGE fcs1_fcs2_cntrl
  fcs_1_2666_2629 ->
  fcs_2_2665_2628
    PROPERTY id = 2662
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 6
    PROPERTY label_y_offset =  - 6
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "512 337 514 464 "

EDGE fcs1_fcs2_health
  fcs_1_2666_2629 ->
  fcs_2_2665_2628
    PROPERTY id = 2663
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 5
    PROPERTY label_y_offset = 22
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "544 338 552 405 542 473 "

EDGE fcs1_fcs2_critical_data
  fcs_1_2666_2629 ->
  fcs_2_2665_2628
    PROPERTY id = 2664
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 11
    PROPERTY label_y_offset = 54
```

```
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "575 348 589 409 562 475 "

EDGE fcs_dss_tgt_data
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2836
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 211
    PROPERTY label_y_offset = 23
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "581 338 619 362 692 366 "

EDGE fcs_nss_cmd
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2837
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 253
    PROPERTY label_y_offset =  - 7
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "518 128 589 101 684 99 "

EDGE fcs_dss_cmd
  fcs_1_2666_2629 ->
  EXTERNAL
    PROPERTY id = 2838
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 284
    PROPERTY label_y_offset =  - 17
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "498 113 557 72 692 69 "

EDGE fcs_dss_tgt_data
  fcs_2_2665_2628 ->
  EXTERNAL
    PROPERTY id = 2839
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 208
    PROPERTY label_y_offset = 19
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "591 688 638 707 696 705 "

  EDGE fcs_dss_cmd
    fcs_2_2665_2628 ->
    EXTERNAL
      PROPERTY id = 2840
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 213
      PROPERTY label_y_offset = 17
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "585 705 608 721 637 731 701 728 "

  EDGE fcs_nss_cmd
    fcs_2_2665_2628 ->
    EXTERNAL
      PROPERTY id = 2841
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 223
      PROPERTY label_y_offset = 12
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "572 735 599 748 624 754 700 751 "

  EDGE fcs_wss_rel_cmd
    fcs_1_2666_2629 ->
    EXTERNAL
      PROPERTY id = 2792
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 205
      PROPERTY label_y_offset = 23
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "577 265 634 290 683 295 "

  EDGE fcs_wss_rel_cmd
    fcs_2_2665_2628 ->
    EXTERNAL
      PROPERTY id = 2793
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 205
      PROPERTY label_y_offset = 14
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 15
      PROPERTY spline = "594 640 630 649 688 651 "

DATA STREAM
  fcs2_fcs1_critical_data : fcs_critical_format,
  fcs2_fcs1_health : fcs_health_format,
  fcs_2_fcs1_cntrl : fcs_cntrl_format
CONTROL CONSTRAINTS
```

```
      OPERATOR fcs_2_2665_2628
        PERIOD 20 MS

      OPERATOR fcs_1_2666_2629
        PERIOD 20 MS
END

OPERATOR mass_storage_subsystem_2581
SPECIFICATION
    INPUT
      fcs_mss_cmd : fcs_status_format
    OUTPUT
      mss_wss_stores_data : stores_format
    OUTPUT
      mss_fcs_msn_data : msn_format
    OUTPUT
      mss_wss_wpn_data : BOOLEAN
    MAXIMUM EXECUTION TIME 1 MS
END
IMPLEMENTATION
  GRAPH
    VERTEX mss_mode_control_2690_2680 : 50 MICROSEC
      PROPERTY x = 151
      PROPERTY y = 38
      PROPERTY radius = 71
      PROPERTY color = 28
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 45
      PROPERTY label_y_offset = 92
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 125
      PROPERTY met_y_offset =  - 5
      PROPERTY is_terminator = FALSE

    VERTEX mss_wpn_data_processing_2691_2681 : 100 MICROSEC
      PROPERTY x = 394
      PROPERTY y = 404
      PROPERTY radius = 71
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 28
      PROPERTY label_y_offset = 103
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 125
      PROPERTY met_y_offset =  - 5
      PROPERTY is_terminator = FALSE

    VERTEX mss_stores_data_processing_2692_2682 : 100 MICROSEC
      PROPERTY x = 396
      PROPERTY y = 585
      PROPERTY radius = 71
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 30
      PROPERTY label_y_offset = 104
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 121
      PROPERTY met_y_offset =  - 5
      PROPERTY is_terminator = FALSE

    VERTEX mss_msn_data_processing_2802_2799 : 100 MICROSEC
      PROPERTY x = 395
      PROPERTY y = 191
      PROPERTY radius = 71
      PROPERTY color = 62
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 28
      PROPERTY label_y_offset = 103
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 121
      PROPERTY met_y_offset =  - 5
      PROPERTY is_terminator = FALSE

    EDGE mss_send_wpn_data
      mss_mode_control_2690_2680 ->
      mss_wpn_data_processing_2691_2681
        PROPERTY id = 2685
        PROPERTY label_font = 2
        PROPERTY label_x_offset = 56
        PROPERTY label_y_offset = 51
        PROPERTY latency_font = 3
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 0
        PROPERTY spline = "247 273 325 396 "

    EDGE mss_send_stores_data
      mss_mode_control_2690_2680 ->
      mss_stores_data_processing_2692_2682
        PROPERTY id = 2686
        PROPERTY label_font = 2
        PROPERTY label_x_offset =  - 49
        PROPERTY label_y_offset = 139
        PROPERTY latency_font = 3
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 0
        PROPERTY spline = "223 289 270 451 364 571 "

    EDGE mss_wss_wpn_data
      mss_wpn_data_processing_2691_2681 ->
      EXTERNAL
        PROPERTY id = 2688
        PROPERTY label_font = 2
        PROPERTY label_x_offset = 112
        PROPERTY label_y_offset =  - 10
        PROPERTY latency_font = 3
```

```
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "545 471 818 471 "

  EDGE mss_wss_stores_data
    mss_stores_data_processing_2692_2682 ->
    EXTERNAL
      PROPERTY id = 2689
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 78
      PROPERTY label_y_offset =  - 11
      PROPERTY latency_font = 3
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "556 652 814 652 "

  EDGE mss_send_msn_data
    mss_mode_control_2690_2680 ->
    mss_msn_data_processing_2802_2799
      PROPERTY id = 2800
      PROPERTY label_font = 2
      PROPERTY label_x_offset =  - 45
      PROPERTY label_y_offset =  - 56
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "303 206 364 259 "

  EDGE mss_fcs_msn_data
    mss_msn_data_processing_2802_2799 ->
    EXTERNAL
      PROPERTY id = 2801
      PROPERTY label_font = 2
      PROPERTY label_x_offset = 112
      PROPERTY label_y_offset =  - 11
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "556 259 830 260 "

  EDGE fcs_mss_cmd
    EXTERNAL ->
    mss_mode_control_2690_2680
      PROPERTY id = 2835
      PROPERTY label_font = 2
      PROPERTY label_x_offset =  - 92
      PROPERTY label_y_offset =  - 16
      PROPERTY latency_font = 2
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 0
      PROPERTY spline = "54 183 126 182 "

DATA STREAM
    mss_send_msn_data : BOOLEAN,
    mss_send_stores_data : BOOLEAN,
    mss_send_wpn_data : BOOLEAN
  CONTROL CONSTRAINTS

    OPERATOR mss_mode_control_2690_2680
      MINIMUM CALLING PERIOD 20 MS

    OPERATOR mss_wpn_data_processing_2691_2681
      MINIMUM CALLING PERIOD 20 MS

    OPERATOR mss_stores_data_processing_2692_2682
      MINIMUM CALLING PERIOD 20 MS

    OPERATOR mss_msn_data_processing_2802_2799
      MINIMUM CALLING PERIOD 20 MS
END

OPERATOR mss_mode_control_2690
SPECIFICATION
    INPUT
      fcs_mss_cmd : fcs_status_format
    OUTPUT
      mss_send_wpn_data : BOOLEAN
    OUTPUT
      mss_send_stores_data : BOOLEAN
    OUTPUT
      mss_send_msn_data : BOOLEAN
    MAXIMUM EXECUTION TIME 50 MICROSEC
END
IMPLEMENTATION ADA mss_mode_control_2690

END

OPERATOR mss_msn_data_processing_2802
SPECIFICATION
    INPUT
      mss_send_msn_data : BOOLEAN
    OUTPUT
      mss_fcs_msn_data : msn_format
    MAXIMUM EXECUTION TIME 100 MICROSEC
END
IMPLEMENTATION ADA mss_msn_data_processing_2802

END

OPERATOR mss_stores_data_processing_2692
SPECIFICATION
    INPUT
      mss_send_stores_data : BOOLEAN
    OUTPUT
      mss_wss_stores_data : stores_format
    MAXIMUM EXECUTION TIME 100 MICROSEC
```

```
END
IMPLEMENTATION ADA mss_stores_data_processing_2692

END

OPERATOR mss_wpn_data_processing_2691
SPECIFICATION
    INPUT
      mss_send_wpn_data : BOOLEAN
    OUTPUT
      mss_wss_wpn_data : BOOLEAN
    MAXIMUM EXECUTION TIME 100 MICROSEC
END
IMPLEMENTATION ADA mss_wpn_data_processing_2691

END

OPERATOR nav_sub_system_2578
SPECIFICATION
    INPUT
      ownship_data : nav_format
    INPUT
      fcs_nss_cmd : fcs_status_format
    OUTPUT
      nss_fcs_nav_data : nav_format
    MAXIMUM EXECUTION TIME 1 MS
END
IMPLEMENTATION ADA nav_sub_system_2578

END

OPERATOR rack_interface_unit_2767
SPECIFICATION
    INPUT
      wpn_release_signal : wpn_id_format
    OUTPUT
      hung_weapon : exception_type
    MAXIMUM EXECUTION TIME 100 MICROSEC
    KEYWORDS
      random_process, hung_exception
    DESCRIPTION
      {This interface is responsible for releasing the weapon id provided
       in wpn_release_signal.  There are no provisions for detecting the
       multiple release of a weapon.  An exception is generated, on a
       random basis, to report a hung weapon.  It is possible to attempt
       another release of the hung weapon by sending the release command
       again for that weapon.  If hung, weapons behind the hung weapon can
       not be released. }
END
IMPLEMENTATION Ada rack_interface_unit_2767

END

OPERATOR radar_sub_system_2577
SPECIFICATION
    INPUT
      ownship_data : nav_format
    INPUT
      sim_grd_tgt_data : nav_format
    INPUT
      sim_air_tgt_data : nav_format
    INPUT
      fcs_rss_cmd : fcs_status_format
    INPUT
      fcs_rss_nav_data : nav_format
    OUTPUT
      rss_fcs_grd_tgt_data : delta_tgt_format
    OUTPUT
      rss_fcs_air_tgt_data : delta_tgt_format
    OUTPUT
      rss_fcs_platform_data : nav_format
    STATES
      rss_mode_phase : Rate_25Hz
      INITIALLY
        0
    MAXIMUM EXECUTION TIME 3 MS
END
IMPLEMENTATION
  GRAPH
    VERTEX rss_mode_control_2607_2585 : 50 MICROSEC
      PROPERTY x = 156
      PROPERTY y = 54
      PROPERTY radius = 64
      PROPERTY color = 28
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 31
      PROPERTY label_y_offset = 84
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 96
      PROPERTY met_y_offset = 0
      PROPERTY is_terminator = FALSE

    VERTEX rss_platform_processing_2608_2586 : 50 MICROSEC
      PROPERTY x = 679
      PROPERTY y = 582
      PROPERTY radius = 62
      PROPERTY color = 63
      PROPERTY label_font = 3
      PROPERTY label_x_offset = 18
      PROPERTY label_y_offset = 86
      PROPERTY met_font = 2
      PROPERTY met_x_offset = 107
      PROPERTY met_y_offset =  - 5
      PROPERTY is_terminator = FALSE

    VERTEX rss_air_air_mode_2609_2587 : 200 MICROSEC
```

```
  PROPERTY x = 674
  PROPERTY y = 406
  PROPERTY radius = 65
  PROPERTY color = 62
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 49
  PROPERTY label_y_offset = 97
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 109
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX rss_air_grd_mode_2610_2588 : 200 MICROSEC
  PROPERTY x = 665
  PROPERTY y = 210
  PROPERTY radius = 66
  PROPERTY color = 62
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 51
  PROPERTY label_y_offset = 95
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 111
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX rss_nav_processing_2611_2589 : 50 MICROSEC
  PROPERTY x = 447
  PROPERTY y = 49
  PROPERTY radius = 67
  PROPERTY color = 63
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 22
  PROPERTY label_y_offset = 89
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 117
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

EDGE fcs_rss_cmd
  EXTERNAL ->
  rss_mode_control_2607_2585
    PROPERTY id = 2590
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 38
    PROPERTY label_y_offset =  - 12
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "21 108 48 86 124 86 "

EDGE fcs_rss_nav_data
  EXTERNAL ->
  rss_nav_processing_2611_2589
    PROPERTY id = 2591
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 59
    PROPERTY label_y_offset =  - 6
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "314 49 342 25 431 25 "

EDGE rss_fcs_grd_tgt_data
  rss_air_grd_mode_2610_2588 ->
  EXTERNAL
    PROPERTY id = 2592
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 56
    PROPERTY label_y_offset =  - 18
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "870 272 881 272 "

EDGE rss_fcs_air_tgt_data
  rss_air_air_mode_2609_2587 ->
  EXTERNAL
    PROPERTY id = 2593
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 3
    PROPERTY label_y_offset =  - 27
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "816 463 890 461 "

EDGE rss_fcs_platform_data
  rss_platform_processing_2608_2586 ->
  EXTERNAL
    PROPERTY id = 2595
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 5
    PROPERTY label_y_offset =  - 24
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "813 637 873 636 "

EDGE rss_mode_phase
  rss_mode_control_2607_2585 ->
  rss_mode_control_2607_2585
    PROPERTY id = 2596
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 40
    PROPERTY label_y_offset = 35
    PROPERTY latency_font = 2
```

```
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "322 99 348 120 326 144 "

EDGE rss_nav_data
  rss_nav_processing_2611_2589 ->
  rss_air_grd_mode_2610_2588
    PROPERTY id = 2598
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 13
    PROPERTY label_y_offset =  - 29
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "547 210 612 260 "

EDGE rss_nav_data
  rss_nav_processing_2611_2589 ->
  rss_air_air_mode_2609_2587
    PROPERTY id = 2599
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 50
    PROPERTY label_y_offset = 54
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "528 256 601 394 "

EDGE rss_exec_phase_all
  rss_mode_control_2607_2585 ->
  rss_nav_processing_2611_2589
    PROPERTY id = 2600
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 33
    PROPERTY label_y_offset = 17
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "309 226 447 226 "

EDGE rss_exec_phase_1
  rss_mode_control_2607_2585 ->
  rss_air_grd_mode_2610_2588
    PROPERTY id = 2601
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 72
    PROPERTY label_y_offset = 15
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "287 242 414 299 600 299 "

EDGE rss_exec_phase_2
  rss_mode_control_2607_2585 ->
  rss_air_air_mode_2609_2587
    PROPERTY id = 2602
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 68
    PROPERTY label_y_offset = 8
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "290 335 444 461 646 468 "

EDGE rss_exec_phase_all
  rss_mode_control_2607_2585 ->
  rss_platform_processing_2608_2586
    PROPERTY id = 2603
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 144
    PROPERTY label_y_offset =  - 3
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "260 386 413 562 559 639 "

EDGE ownship_data
  EXTERNAL ->
  rss_platform_processing_2608_2586
    PROPERTY id = 2604
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 30
    PROPERTY label_y_offset = 26
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "439 711 524 674 619 665 "

EDGE sim_air_tgt_data
  EXTERNAL ->
  rss_air_air_mode_2609_2587
    PROPERTY id = 2605
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 27
    PROPERTY label_y_offset = 24
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 0
    PROPERTY spline = "454 520 496 495 617 498 "

EDGE sim_grd_tgt_data
  EXTERNAL ->
  rss_air_grd_mode_2610_2588
    PROPERTY id = 2606
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 123
```

```
        PROPERTY label_y_offset = 63
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = 0
        PROPERTY spline = "476 346 524 317 620 315 "

  DATA STREAM
    rss_exec_phase_2 : BOOLEAN,
    rss_exec_phase_1 : BOOLEAN,
    rss_exec_phase_all : BOOLEAN,
    rss_nav_data : nav_format
  CONTROL CONSTRAINTS

      OPERATOR rss_mode_control_2607_2585
        TRIGGERED BY ALL
          fcs_rss_cmd
        MINIMUM CALLING PERIOD 20 MS
        MAXIMUM RESPONSE TIME 5 MS
        OUTPUT
          rss_exec_phase_1
          IF rss_mode_phase = 0

        OUTPUT
          rss_exec_phase_2
          IF rss_mode_phase = 1

        OUTPUT
          rss_exec_phase_all
          IF TRUE

      OPERATOR rss_platform_processing_2608_2586
        TRIGGERED BY ALL
          rss_exec_phase_all
        MINIMUM CALLING PERIOD 20 MS

      OPERATOR rss_air_air_mode_2609_2587
        TRIGGERED BY ALL
          rss_exec_phase_2
        MINIMUM CALLING PERIOD 20 MS

      OPERATOR rss_air_grd_mode_2610_2588
        TRIGGERED BY ALL
          rss_exec_phase_1
        MINIMUM CALLING PERIOD 20 MS

      OPERATOR rss_nav_processing_2611_2589
        TRIGGERED BY ALL
          rss_exec_phase_all
        MINIMUM CALLING PERIOD 20 MS
END

OPERATOR rss_air_air_mode_2609
SPECIFICATION
    INPUT
      rss_nav_data : nav_format
    INPUT
      rss_exec_phase_2 : BOOLEAN
    INPUT
      sim_air_tgt_data : nav_format
    OUTPUT
      rss_fcs_air_tgt_data : delta_tgt_format
    MAXIMUM EXECUTION TIME 200 MICROSEC
END
IMPLEMENTATION ADA rss_air_air_mode_2609

END

OPERATOR rss_air_grd_mode_2610
SPECIFICATION
    INPUT
      rss_nav_data : nav_format
    INPUT
      rss_exec_phase_1 : BOOLEAN
    INPUT
      sim_grd_tgt_data : nav_format
    OUTPUT
      rss_fcs_grd_tgt_data : delta_tgt_format
    MAXIMUM EXECUTION TIME 200 MICROSEC
    KEYWORDS
      type_processing
    DESCRIPTION
      {Calculate delta x, y, z for target from ownship position.
       Calculate range based on delta x, y, z }
    AXIOMS
      {delta_x = tgt_x - own_x
       delta_y = tgt_y - own_y
       delta_z = tgt_z - own_z
       range = sqrt(delta_x*delta_x + delta_y*delta_y + delta_z*delta_z) }
END
IMPLEMENTATION ADA rss_air_grd_mode_2610

END

OPERATOR rss_mode_control_2607
SPECIFICATION
    INPUT
      fcs_rss_cmd : fcs_status_format
    INPUT
      rss_mode_phase : Rate_25Hz
    OUTPUT
      rss_mode_phase : Rate_25Hz
    OUTPUT
      rss_exec_phase_all : BOOLEAN
    OUTPUT
      rss_exec_phase_1 : BOOLEAN
    OUTPUT
```

```
      rss_exec_phase_2 : BOOLEAN
    MAXIMUM EXECUTION TIME 50 MICROSEC
    KEYWORDS
      Periodic, Scheduling, Rate, Phase
    DESCRIPTION
      {This operator is responsible for scheduling all radar processes.
       Processes are scheduled at either 50Hz or 25Hz.  For those
       processes that are scheduled at 25Hz, they can be executed in
       either Phase 1 or Phase 2. }
    AXIOMS
      {rss_exec_phase_all AND (rss_exec_phase_1 OR rss_exec_phase_2) = true
       rss_exec_phase_1 XOR rss_exeec_phase_2 = true }
END
IMPLEMENTATION ADA rss_mode_control_2607

END

OPERATOR rss_nav_processing_2611
SPECIFICATION
    INPUT
      fcs_rss_nav_data : nav_format
    INPUT
      rss_exec_phase_all : BOOLEAN
    OUTPUT
      rss_nav_data : nav_format
    MAXIMUM EXECUTION TIME 50 MICROSEC
    KEYWORDS
      pass_thru
    DESCRIPTION
      {Pass-thru of fcs_rss_nav_data }
END
IMPLEMENTATION ADA rss_nav_processing_2611

END

OPERATOR rss_platform_processing_2608
SPECIFICATION
    INPUT
      rss_exec_phase_all : BOOLEAN
    INPUT
      ownship_data : nav_format
    OUTPUT
      rss_fcs_platform_data : nav_format
    MAXIMUM EXECUTION TIME 50 MICROSEC
    KEYWORDS
      pass_thru
    DESCRIPTION
      {Pass thru of ownship_data.
       Removed error bias from design. }
END
IMPLEMENTATION ADA rss_platform_processing_2608

END
```

```
OPERATOR weapon_sub_system_2579
SPECIFICATION
    INPUT
      fcs_wss_cmd : fcs_status_format
    INPUT
      mss_wss_stores_data : stores_format
    INPUT
      av_consent_switch : BOOLEAN
    INPUT
      fcs_wss_rel_cmd : BOOLEAN
    INPUT
      mss_wss_wpn_data : BOOLEAN
    OUTPUT
      wss_fcs_wpn_data : wpn_format
    STATES
      wpn_count : INTEGER
      INITIALLY
        0
    STATES
      wpn_id : wpn_id_format
      INITIALLY
        no_weapon_selected
    STATES
      next_wpn_req : BOOLEAN
      INITIALLY
        FALSE
    STATES
      release_count : INTEGER
      INITIALLY
        0
    STATES
      wpn_release_signal : wpn_id_format
      INITIALLY
        no_weapon_selected
    STATES
      wss_init_request : BOOLEAN
      INITIALLY
        FALSE
    STATES
      wss_init_required : BOOLEAN
      INITIALLY
        FALSE
    MAXIMUM EXECUTION TIME 5 MS
END
IMPLEMENTATION
  GRAPH
    VERTEX rack_interface_unit_2767_2766 : 100 MICROSEC
      PROPERTY x = 702
      PROPERTY y = 598
      PROPERTY radius = 65
      PROPERTY color = 62
      PROPERTY label_font = 3
```

```
  PROPERTY label_x_offset = 26
  PROPERTY label_y_offset = 89
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 134
  PROPERTY met_y_offset = 38
  PROPERTY is_terminator = FALSE

VERTEX wss_initiate_release_squence_2753_2749 : 200 MICROSEC
  PROPERTY x = 359
  PROPERTY y = 372
  PROPERTY radius = 64
  PROPERTY color = 62
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 29
  PROPERTY label_y_offset = 95
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 112
  PROPERTY met_y_offset = 9
  PROPERTY is_terminator = FALSE

VERTEX wss_mode_control_2754_2750 : 50 MICROSEC
  PROPERTY x = 133
  PROPERTY y = 67
  PROPERTY radius = 63
  PROPERTY color = 28
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 39
  PROPERTY label_y_offset = 85
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 109
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX wss_inventory_management_2755_2751 : 200 MICROSEC
  PROPERTY x = 359
  PROPERTY y = 164
  PROPERTY radius = 64
  PROPERTY color = 62
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 22
  PROPERTY label_y_offset = 88
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 107
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX wss_not_implemented_2789_2788
  PROPERTY x = 367
  PROPERTY y = 611
  PROPERTY radius = 62
  PROPERTY color = 46
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 15
  PROPERTY label_y_offset = 83
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 124
  PROPERTY met_y_offset =  - 5
  PROPERTY is_terminator = FALSE

VERTEX wss_release_processing_2756_2752 : 200 MICROSEC
  PROPERTY x = 689
  PROPERTY y = 374
  PROPERTY radius = 63
  PROPERTY color = 62
  PROPERTY label_font = 3
  PROPERTY label_x_offset = 22
  PROPERTY label_y_offset = 83
  PROPERTY met_font = 2
  PROPERTY met_x_offset = 134
  PROPERTY met_y_offset = 33
  PROPERTY is_terminator = FALSE

EDGE fcs_wss_cmd
  EXTERNAL ->
  wss_mode_control_2754_2750
    PROPERTY id = 2768
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 47
    PROPERTY label_y_offset =  - 6
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "24 108 68 85 "

EDGE wss_init_required
  wss_mode_control_2754_2750 ->
  wss_inventory_management_2755_2751
    PROPERTY id = 2769
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 31
    PROPERTY label_y_offset =  - 13
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "333 134 "

EDGE wss_init_request
  wss_inventory_management_2755_2751 ->
  wss_mode_control_2754_2750
    PROPERTY id = 2770
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 50
    PROPERTY label_y_offset = 25
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
```

```
    PROPERTY spline = "292 208 "

EDGE mss_wss_stores_data
  EXTERNAL ->
  wss_inventory_management_2755_2751
    PROPERTY id = 2771
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 256
    PROPERTY label_y_offset =  - 18
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "39 258 224 258 326 258 "

EDGE wss_exec_phase
  wss_mode_control_2754_2750 ->
  wss_initiate_release_squence_2753_2749
    PROPERTY id = 2772
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 112
    PROPERTY label_y_offset = 6
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "199 233 204 345 266 402 343 433 "

EDGE fcs_wss_rel_cmd
  EXTERNAL ->
  wss_initiate_release_squence_2753_2749
    PROPERTY id = 2773
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 262
    PROPERTY label_y_offset =  - 24
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "42 472 146 474 276 477 341 472 "

EDGE wpn_release_signal : 20 MS
  wss_release_processing_2756_2752 ->
  rack_interface_unit_2767_2766
    PROPERTY id = 2774
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 131
    PROPERTY label_y_offset = 29
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset =  - 77
    PROPERTY latency_y_offset = 49
    PROPERTY spline = "722 553 "

EDGE hung_weapon : 20 MS
  rack_interface_unit_2767_2766 ->
  wss_release_processing_2756_2752
    PROPERTY id = 2775
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 2
    PROPERTY label_y_offset =  - 8
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "810 561 "

EDGE release_count
  wss_initiate_release_squence_2753_2749 ->
  wss_release_processing_2756_2752
    PROPERTY id = 2776
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 37
    PROPERTY label_y_offset = 24
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "543 403 636 403 "

EDGE hung_weapon
  wss_release_processing_2756_2752 ->
  wss_initiate_release_squence_2753_2749
    PROPERTY id = 2777
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 36
    PROPERTY label_y_offset = 23
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "661 484 536 478 "

EDGE next_wpn_req
  wss_release_processing_2756_2752 ->
  wss_inventory_management_2755_2751
    PROPERTY id = 2778
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 176
    PROPERTY label_y_offset =  - 42
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "657 388 572 361 "

EDGE wpn_id
  wss_inventory_management_2755_2751 ->
  wss_release_processing_2756_2752
    PROPERTY id = 2779
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
```

```
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "515 265 611 313 684 370 "

EDGE hung_weapon
  wss_release_processing_2756_2752 ->
  wss_inventory_management_2755_2751
    PROPERTY id = 2780
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "718 345 653 280 553 246 "

EDGE wss_exec_phase
  wss_mode_control_2754_2750 ->
  wss_inventory_management_2755_2751
    PROPERTY id = 2781
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 10
    PROPERTY label_y_offset = 36
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "211 222 226 275 293 300 "

EDGE wss_fcs_wpn_data
  wss_inventory_management_2755_2751 ->
  EXTERNAL
    PROPERTY id = 2782
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 226
    PROPERTY label_y_offset =  - 21
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "512 176 567 161 860 166 "

EDGE wpn_count
  wss_release_processing_2756_2752 ->
  wss_release_processing_2756_2752
    PROPERTY id = 2783
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "769 358 810 320 819 372 "

EDGE av_consent_switch
  EXTERNAL ->
  wss_release_processing_2756_2752
    PROPERTY id = 2784
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 503
    PROPERTY label_y_offset = 9
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "22 580 395 585 515 559 625 536 "

EDGE mss_wss_wpn_data
  EXTERNAL ->
  wss_not_implemented_2789_2788
    PROPERTY id = 2790
    PROPERTY label_font = 2
    PROPERTY label_x_offset =  - 233
    PROPERTY label_y_offset =  - 19
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "39 675 260 675 "

DATA STREAM
  hung_weapon : exception_type,
  wss_exec_phase : BOOLEAN
CONTROL CONSTRAINTS

  OPERATOR rack_interface_unit_2767_2766
    TRIGGERED BY ALL
      wpn_release_signal
    MINIMUM CALLING PERIOD 20 MS
    EXCEPTION hung_weapon

  OPERATOR wss_initiate_release_squence_2753_2749
    TRIGGERED BY SOME
      wss_exec_phase, hung_weapon
    MINIMUM CALLING PERIOD 20 MS

  OPERATOR wss_mode_control_2754_2750
    TRIGGERED BY ALL
      fcs_wss_cmd
    MINIMUM CALLING PERIOD 20 MS

  OPERATOR wss_inventory_management_2755_2751
    TRIGGERED BY SOME
      wss_init_required, wss_exec_phase, next_wpn_req, hung_weapon
    MINIMUM CALLING PERIOD 20 MS

  OPERATOR wss_not_implemented_2789_2788
    TRIGGERED IF
      FALSE
```

```
      OPERATOR wss_release_processing_2756_2752
        TRIGGERED BY SOME
          release_count, hung_weapon
        MINIMUM CALLING PERIOD 20 MS
END

OPERATOR wss_initiate_release_squence_2753
SPECIFICATION
    INPUT
      wss_exec_phase : BOOLEAN
    INPUT
      fcs_wss_rel_cmd : BOOLEAN
    INPUT
      hung_weapon : exception_type
    OUTPUT
      release_count : INTEGER
    MAXIMUM EXECUTION TIME 200 MICROSEC
END
IMPLEMENTATION ADA wss_initiate_release_squence_2753

END

OPERATOR wss_inventory_management_2755
SPECIFICATION
    INPUT
      wss_init_required : BOOLEAN
    INPUT
      mss_wss_stores_data : stores_format
    INPUT
      next_wpn_req : BOOLEAN
    INPUT
      hung_weapon : exception_type
    INPUT
      wss_exec_phase : BOOLEAN
    OUTPUT
      wss_init_request : BOOLEAN
    OUTPUT
      wpn_id : wpn_id_format
    OUTPUT
      wss_fcs_wpn_data : wpn_format
    MAXIMUM EXECUTION TIME 200 MICROSEC
END
IMPLEMENTATION ADA wss_inventory_management_2755

END

OPERATOR wss_mode_control_2754
SPECIFICATION
    INPUT
      fcs_wss_cmd : fcs_status_format
    INPUT
      wss_init_request : BOOLEAN
    OUTPUT
      wss_init_required : BOOLEAN
    OUTPUT
      wss_exec_phase : BOOLEAN
    MAXIMUM EXECUTION TIME 50 MICROSEC
END
IMPLEMENTATION ADA wss_mode_control_2754

END

OPERATOR wss_not_implemented_27B9
SPECIFICATION
    INPUT
      mss_wss_wpn_data : BOOLEAN
    KEYWORDS
      not_implemented
    DESCRIPTION
      {This portion of the avionics model is not implemented.  However,
       an operator is needed to receive the input stream. }
END
IMPLEMENTATION Ada wss_not_implemented_27B9

END

OPERATOR wss_release_processing_2756
SPECIFICATION
    INPUT
      hung_weapon : exception_type
    INPUT
      release_count : INTEGER
    INPUT
      wpn_id : wpn_id_format
    INPUT
      wpn_count : INTEGER
    INPUT
      av_consent_switch : BOOLEAN
    OUTPUT
      wpn_release_signal : wpn_id_format
    OUTPUT
      hung_weapon : exception_type
    OUTPUT
      next_wpn_req : BOOLEAN
    OUTPUT
      wpn_count : INTEGER
    MAXIMUM EXECUTION TIME 200 MICROSEC
END
IMPLEMENTATION ADA wss_release_processing_2756

END
```

APPENDIX C. PSDL UNIQUE SUFFIX NAMING CONVENTION MEMO

```
From berzins@cs.nps.navy.mil Thu Jul 25 15:40:50 1996
Return-Path: <berzins@cs.nps.navy.mil>
Received: from suns6.cs.nps.navy.mil by cs.nps.navy.mil (4.1/SMI-4.1)
id AA10627; Thu, 25 Jul 96 15:40:50 PDT
From: berzins@cs.nps.navy.mil (valdis Berzins)
Received: by suns6.cs.nps.navy.mil (4.1) id AA08716; Thu,
                                        25 Jul 96 15:40:43 PDT
Date: Thu, 25 Jul 96 15:40:43 PDT
Message-Id: <9607252240.AA08716@suns6.cs.nps.navy.mil>
To: berzins@cs.nps.navy.mil, irwin@cs.nps.navy.mil,
        kbmoelle@cs.nps.navy.mil,
        luqi@cs.nps.navy.mil, mantak@cs.nps.navy.mil,
        senrique@cs.nps.navy.mil
Subject: explanation of the new unique suffix conventions
Status: RO

Unique integer suffixes in CAPS Release 2

Valdis Berzins, 7/25/96

To properly implement PSDL scope rules,
which say that the names of the children of a
composite bubble are local to that bubble,
we need unique internal names for PSDL operators.
To allow multiple instances of the same type operation,
we also need unique internal names for graph nodes.

The implementation in CAPS Release 2 will be based on the
following conventions:

1. The psdl editor will supply unique internal integer suffixes
for the names of stand-alone PSDL operators.
These suffixes WILL NOT BE VISIBLE TO THE USER OF THE EDITOR,
and will be supplied automatically in generated skeleton code
for operators with IMPLEMENTATION ADA.
The purpose of these suffixes is to allow two different
composite bubbles to have children of the same name without
conflicts between the Ada module names.
```

2. Names of PSDL data types will not have automatically created integer suffixes, because they are not needed.

3. Names of operations of PSDL data types will not have automatically created integer suffixes.
This implies the translator does not have to resolve overloaded operators.

4. The psdl editor will add a unique integer suffix to each operation name to get the name of the corresponding graph node. This enables multiple instances of an operation of a data type to consistently co-exist in the same graph.

5. In release 2 the integer suffixes will be unique throughout the entire prototype and will be obtained from a procedure local to the editors (because there will be no DDB in release 2).
In future releases the suffixes will be unique across all prototypes developed at a given installation, and will be obtained from a unique id server in the DDB.

Some consequences of these conventions:

A. The name of the ada operation can be recovered from the name of the graph node by removing a trailing numeric suffix of the form ("_" digit+).
Note that the name of the graph node does not include the optional parameter binding part.

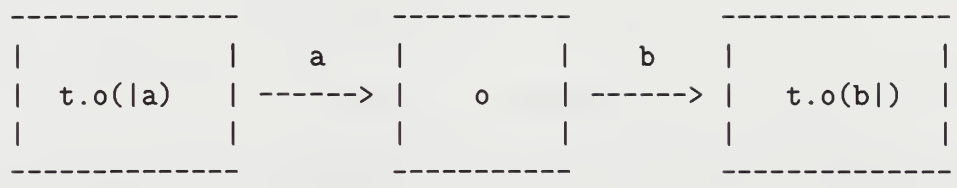
B. For prototypes created under caps release 2, graph nodes corresponding to stand-alone operators will have two integer suffixes, but only the second one need be recognized by computations in translation. The schedulers are only concerned with graph node ids, and hence are not affected by this convention.

C. Graph nodes corresponding to operations of a PSDL data type have a single suffix. This suffix enables multiple instances of a type operation to appear in a graph without danger of confusion.

D. Attributes and control constraints are located by visual navigation through the graphic editor, to avoid confusing the correspondence between graph nodes and control constraints.

```
E. PSDL files created under release 1 may not have these suffixes.
The concrete parsing rules of the sde should recognize such cases,
and should add the needed suffixes if they are missing,
to enable the sde to read old psdl files (CAPS release 1 format).

EXAMPLE

Displayed form of the psdl, with explanatory comments:

OPERATOR o  -- internal name o_13, graph node o_13_20
  SPECIFICATION
    INPUT a: boolean
    OUTPUT b: integer
  END
  IMPLEMENTATION ADA END

TYPE t  -- internal name t
  OPERATOR o  -- overloaded, internal name o, graph node t.o_21(b|)
    SPECIFICATION
      INPUT x: integer
    END
  OPERATOR o  -- overloaded, internal name o, graph node t.o_22(|a)
    SPECIFICATION
      OUTPUT y: boolean
    END
  END
  IMPLEMENTATION ADA END

OPERATOR top
  SPECIFICATION
  END
  IMPLEMENTATION
    GRAPH  -- internal form of the graph, whose display is suppressed:
      VERTEX o_13_20
      VERTEX t.o_21(b|)
      VERTEX t.o_22(|a)
      EDGE a t.o_22(|a) -> o_13_20
      EDGE b o_13_20 -> t.o_21(b|)
  END
```

Displayed form of the graph:



APPENDIX D. GRAPH EDITOR PROGRAM SOURCE CODE

The following is the complete source code listing of the graph editor portion of the PSDL Editor. The original graph editor source code was developed by Captain Robert Dixon, USMC [Ref. 13]. Additional work was performed by Mr. Douglas Lange and Mr. Dagohoy Anunciado as part of the NPS CS4520 (AY96Q4) class project. In several of the graph editor files, the authors of the graph editor have adapted code written by others. Such instances are credited within the source files.

For a starting point, `ge_driver.C` contains the `main()` routine for the graph editor. This routine is called by the syntax-directed editor to start the graph editor. After initialization and establishing communications with the syntax-directed editor, the routine `edit_graph()` is called, which can be found at the end of the file `graph_editor.C`.

```
.SUFFIXES: .o .C .c .h

##################
#C = cc
#CC = CC
#LIBS = -lXm -lXt -lXext -lX11 -lm
##################
CC   = g++
C    = gcc
LIBS = -lXm -lXt -lXext -lX11 -lm  -g++ -gcc
##################

DEBUG = -g -DGE_DEBUG
CFLAGS = $(DEBUG) -D_NO_PROTO
C++FLAGS = $(DEBUG) $(C++INC) $(C++LIB) -DFUNCPROTO -DXTFUNCPROTO


#LIBS = -lXm -lXt -lXext -lX11 -lm  -gcc /usr/local/lib/libg++.a
#LIBS = -lC -lXm -lXt -lXext -lX11 -lm
#LIBS =  -lXm -lXt -lXext -lX11 -lm -y_close_id_server
#DEBUG = -g
#C++INC = -L /usr/lang/SC3.0.1/lib
#C++LIB = -I /usr/lang/SC3.0.1/include/CC_413_U1
#CFLAGS = $(DEBUG)


all: sde edit_graph

.c.o:
$(C) $(CFLAGS) -c $*.c

.C.o:
$(CC) $(C++FLAGS) -c $*.C


SDEOBJECTS= sde.o sde_simulator.o inter_process_utilities.o ge_utilities.o\
ge_utilities_debug.o

GOBJECTS= graph_editor.o operator_object.o stream_object.o spline_object.o\
graph_object_list.o font_table.o graph_object.o\
setcursor.o gettopshell.o postpopup.o build_option.o\
timer_tool.o action_area.o warning.o ge_utilities.o\
stream_property_menu.o operator_property_menu.o windows.o\
get_unique_id.o \
ge_utilities_debug.o\
ge_driver.o inter_process_utilities.o report_errors.o


sde: $(SDEOBJECTS)
$(C) $(CFLAGS) -o ../sde $(SDEOBJECTS) $(LIBS)

edit_graph: $(GOBJECTS)
$(CC) $(C++FLAGS) -o ../edit_graph $(GOBJECTS) $(LIBS)

clean:
$(RM) $(GOBJECTS)
$(RM) $(SDEOBJECTS)
$(RM) ../sde
$(RM) ../edit_graph

### NOTE:  The following dependencies can be generated using the -MM
###        flag on the gcc and g++ compiler.

action_area.o: action_area.C action_area.h graph_editor.h ge_interface.h \
  windows.h ge_utilities.h
$(CC) $(C++FLAGS) -c action_area.C

build_option.o: build_option.c build_option.h
$(CC) $(C++FLAGS) -c build_option.c

font_table.o: font_table.C font_table.h ge_defs.h ge_interface.h resources.h \
  ge_utilities.h warning.h
$(CC) $(C++FLAGS) -c font_table.C

ge_driver.o: ge_driver.C inter_process_utilities.h ge_interface.h \
  graph_editor.h windows.h ge_utilities.h get_unique_id.h
$(CC) $(C++FLAGS) -c ge_driver.C

ge_utilities.o: ge_utilities.c ge_utilities.h ge_interface.h
$(C) $(CFLAGS) -c ge_utilities.c

ge_utilities_debug.o: ge_utilities_debug.c ge_interface.h \
  ge_interface_labels.h ge_utilities.h ge_utilities_debug.h
$(C) $(CFLAGS) -c ge_utilities_debug.c

get_unique_id.o: get_unique_id.c get_unique_id.h
$(C) $(CFLAGS) -c get_unique_id.c

gettopshell.o: gettopshell.c gettopshell.h
$(CC) $(C++FLAGS) -c gettopshell.c

graph_editor.o: graph_editor.C action_area.h build_option.h ge_defs.h \
  ge_interface.h resources.h gettopshell.h graph_editor.h windows.h \
  ge_utilities.h graph_object_list.h graph_object.h font_table.h \
  operator_object.h postpopup.h setcursor.h spline_object.h stream_object.h \
  operator_property_menu.h stream_property_menu.h timer_tool.h warning.h \
  ge_utilities_debug.h report_errors.h
$(CC) $(C++FLAGS) -c graph_editor.C

graph_object.o: graph_object.C graph_object.h ge_defs.h ge_interface.h \
  resources.h font_table.h
$(CC) $(C++FLAGS) -c graph_object.C

graph_object_list.o: graph_object_list.C graph_object_list.h ge_defs.h \
  ge_interface.h resources.h ge_utilities.h graph_object.h font_table.h \
  operator_object.h stream_object.h spline_object.h warning.h \
```

```
  ge_utilities_debug.h get_unique_id.h
$(CC) $(C++FLAGS) -c graph_object_list.C

inter_process_utilities.o: inter_process_utilities.c \
  inter_process_utilities.h ge_interface.h ge_utilities.h
$(C) $(CFLAGS) -c inter_process_utilities.c

operator_object.o: operator_object.C operator_object.h ge_defs.h \
  ge_interface.h resources.h graph_object.h font_table.h ge_utilities.h
$(CC) $(C++FLAGS) -c operator_object.C

operator_property_menu.o: operator_property_menu.C ge_defs.h ge_interface.h \
  resources.h ge_utilities.h gettopshell.h graph_editor.h windows.h \
  graph_object_list.h graph_object.h font_table.h \
  operator_object.h operator_property_menu.h warning.h action_area.h \
  build_option.h
$(CC) $(C++FLAGS) -c operator_property_menu.C

postpopup.o: postpopup.c gettopshell.h setcursor.h
$(CC) $(C++FLAGS) -c postpopup.c

report_errors.o: report_errors.C ge_interface.h graph_editor.h windows.h \
  ge_utilities.h warning.h action_area.h
$(CC) $(C++FLAGS) -c report_errors.C

sde.o: sde.c inter_process_utilities.h ge_interface.h sde_simulator.h
$(C) $(CFLAGS) -c sde.c

sde_simulator.o: sde_simulator.c ge_utilities.h ge_interface.h \
  sde_simulator.h inter_process_utilities.h
$(C) $(CFLAGS) -c sde_simulator.c

setcursor.o: setcursor.c setcursor.h
$(CC) $(C++FLAGS) -c setcursor.c

spline_object.o: spline_object.C ge_defs.h ge_interface.h resources.h \
  spline_object.h stream_object.h graph_object.h font_table.h \
  operator_object.h ge_utilities.h warning.h
$(CC) $(C++FLAGS) -c spline_object.C

stream_object.o: stream_object.C stream_object.h ge_defs.h ge_interface.h \
  resources.h graph_object.h font_table.h spline_object.h operator_object.h \
  ge_utilities.h graph_object_list.h
$(CC) $(C++FLAGS) -c stream_object.C

stream_property_menu.o: stream_property_menu.C ge_defs.h ge_interface.h \
  resources.h ge_utilities.h gettopshell.h graph_editor.h windows.h \
  graph_object_list.h graph_object.h font_table.h \
  operator_object.h spline_object.h stream_object.h warning.h
$(CC) $(C++FLAGS) -c stream_property_menu.C

timer_tool.o: timer_tool.C ge_defs.h ge_interface.h resources.h timer_tool.h \
  ge_utilities.h graph_editor.h windows.h action_area.h warning.h
$(CC) $(C++FLAGS) -c timer_tool.C

warning.o: warning.C warning.h
$(CC) $(C++FLAGS) -c warning.C

windows.o: windows.C windows.h ge_utilities.h ge_interface.h warning.h \
  graph_editor.h
$(CC) $(C++FLAGS) -c windows.C
```

```
#include <Xm/DialogS.h>
#include <Xm/PushBG.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
#include <Xm/PanedW.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/TextF.h>

#ifndef ACTION_AREA_H
#define ACTION_AREA_H 1

#define TIGHTNESS 20

typedef struct {
char  *label;
void  (*callback)(Widget, XtPointer, XtPointer);
XtPointer  data;
} ActionAreaItem;

Widget CreateActionArea(Widget, ActionAreaItem*, int);
void close_dialog(Widget, XtPointer, XtPointer);
void clear_pushed(Widget, XtPointer, XtPointer);

#endif
```

```
#include "action_area.h"
#include "graph_editor.h"

void close_dialog(Widget w, XtPointer client_data, XtPointer call_data) {

Widget shell = (Widget)client_data;

clear_status();

XtDestroyWidget(shell);

}


void clear_pushed(Widget w, XtPointer client_data, XtPointer call_data) {

Widget text_w = (Widget)client_data;
XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

XmTextFieldSetString(text_w, "");

}

Widget CreateActionArea(Widget parent, ActionAreaItem *actions, int num_actions) {

Widget  action_area, widget;
int i;

action_area = XtVaCreateWidget("action_area", xmFormWidgetClass, parent,
XmNfractionBase, TIGHTNESS*num_actions -1,
XmNleftOffset, 10,
XmNrightOffset, 10,
NULL);

for(i=0; i<num_actions; i++) {
widget = XtVaCreateManagedWidget(actions[i].label,
xmPushButtonWidgetClass, action_area,
XmNleftAttachment, i ? XmATTACH_POSITION : XmATTACH_FORM,
XmNleftPosition, TIGHTNESS * i,
XmNtopAttachment, XmATTACH_FORM,
XmNbottomAttachment, XmATTACH_FORM,
XmNrightAttachment, i != num_actions - 1 ? XmATTACH_POSITION : XmATTACH_FORM,
XmNrightPosition, TIGHTNESS * i + (TIGHTNESS -1),
XmNshowAsDefault, i == 0,
XmNdefaultButtonShadowThickness, 1,
NULL);
if(actions[i].callback)
XtAddCallback(widget, XmNactivateCallback,
actions[i].callback, actions[i].data);
if(i==0) {
Dimension height, h;
XtVaGetValues(action_area, XmNmarginHeight, &h, NULL);
XtVaGetValues(widget, XmNheight, &height, NULL);
height += 2 * h;
XtVaSetValues(action_area,
XmNdefaultButton, widget,
XmNpaneMaximum, height,
XmNpaneMinimum, height,
NULL);
}

}

XtManageChild(action_area);

return action_area;

}
```

```
/* ************************************************************

Name:     build_option.h
Author:   Lange and Anunciado
Program:  graph_editor
Remarks:

 * Written by Dan Heller.  Copyright 1991, O'Reilly && Associates.
 * This program is freely distributable without licensing fees and
 * is provided without guarantee or warrantee expressed or implied.
 * This program is -not- in the public domain.


 * build_option.h -- Structure used for the final version of BuildMenu().
 * BuildMenu is used to build popup, option, pulldown -and- pullright menus.
 * Menus are defined by declaring an array of MenuItem structures as usual.

History:
  @1   96/09/29  Ken Moeller
       Migration from Motif 1.2 to Motif 1.1.

************************************************************ */

#ifndef BUILD_OPTION_H
#define BUILD_OPTION_H

#include <Xm/MainW.h>
#include <Xm/PanedW.h>
#include <Xm/RowColumn.h>
#include <Xm/DrawingA.h>
#include <Xm/CascadeB.h>
#include <Xm/CascadeBG.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

typedef struct _menu_item {
    char        *label;         /* the label for the item */
    WidgetClass *widget_class;  /* pushbutton, label, separator... */
    Boolean      sensitive;     /* set sensitive (True of False) */
    Boolean      act_display;   /* Set for one to be active */
    char         mnemonic;      /* mnemonic; NULL if none */
    char        *accelerator;   /* accelerator; NULL if none */
    char        *accel_text;    /* to be converted to compound string */
    void       (*callback)();   /* routine to call; NULL if none */
    XtPointer    callback_data; /* client_data for callback() */
    struct _menu_item *subitems; /* pullright menu items, if not NULL */
} MenuItem;

#else

typedef struct _menu_item {
    char        *label;         /* the label for the item */
    WidgetClass *widget_class;  /* pushbutton, label, separator... */
    Boolean      sensitive;     /* set sensitive (True of False) */
    Boolean      act_display;   /* Set for one to be active */
    char         mnemonic;      /* mnemonic; NULL if none */
    char        *accelerator;   /* accelerator; NULL if none */
    char        *accel_text;    /* to be converted to compound string */
    void       (*callback)(Widget, XtPointer, XtPointer); /* routine to call;
                                                             NULL if none */
    XtPointer    callback_data; /* client_data for callback() */
    struct _menu_item *subitems; /* pullright menu items, if not NULL */
} MenuItem;

#endif /* _NO_PROTO */

#ifdef _NO_PROTO

extern Widget BuildMenu() ;

#else

extern Widget BuildMenu(
    Widget parent,
    int menu_type,
    char *menu_title,
    char menu_mnemonic,
    /*    Boolean tear_off, */   /* @1  Not supported in Motif 1.1 */
    MenuItem *items) ;

#endif /* _NO_PROTO */

#ifdef __cplusplus
}  /* Close scope of 'extern "C"' declaration which encloses file. */
#endif

#endif /* BUILD_OPTION_H */
```

```
/* ************************************************************

Name:     op_prop_formal_desc.C
Author:   Lange and Anunciado
Program:  graph_editor
Remarks:

 * Written by Dan Heller.  Copyright 1991, O'Reilly && Associates.
 * This program is freely distributable without licensing fees and
 * is provided without guarantee or warrantee expressed or implied.
 * This program is -not- in the public domain.


 * build_option.c -- The final version of BuildMenu() is used to
 * build popup, option, pulldown -and- pullright menus.  Menus are
 * defined by declaring an array of MenuItem structures as usual.

History:
  @1   96/09/29  Ken Moeller
       Migration from Motif 1.2 to Motif 1.1.

************************************************************ */

#include <Xm/MainW.h>
#include <Xm/PanedW.h>
#include <Xm/RowColumn.h>
#include <Xm/DrawingA.h>
#include <Xm/CascadeBG.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>

#include "build_option.h"

#ifdef __cplusplus
extern "C" {
#endif

/* Build popup, option and pulldown menus, depending on the menu_type.
 * It may be XmMENU_PULLDOWN, XmMENU_OPTION or  XmMENU_POPUP.  Pulldowns
 * return the CascadeButton that pops up the menu.  Popups return the menu.
 * Option menus are created, but the RowColumn that acts as the option
 * "area" is returned unmanaged. (The user must manage it.)
 * Pulldown menus are built from cascade buttons, so this function
 * also builds pullright menus.  The function also adds the right
 * callback for PushButton or ToggleButton menu items.
 */
#ifdef _NO_PROTO

Widget
BuildMenu(parent, menu_type, menu_title, menu_mnemonic, items)
Widget parent;
int menu_type;
char *menu_title, menu_mnemonic;
MenuItem *items;

#else

Widget
BuildMenu(
    Widget parent,
    int menu_type,
    char *menu_title,
    char menu_mnemonic,
    MenuItem *items)

#endif /* _NO_PROTO */
{
    Widget menu, cascade, widget;
    int i;
    XmString str;

    if (menu_type == XmMENU_PULLDOWN || menu_type == XmMENU_OPTION)
        menu = XmCreatePulldownMenu(parent, "_pulldown", NULL, 0);
    else if (menu_type == XmMENU_POPUP)
        menu = XmCreatePopupMenu(parent, "_popup", NULL, 0);
    else {
        XtWarning("Invalid menu type passed to BuildMenu()");
        return NULL;
    }

    /* Pulldown menus require a cascade button to be made */
    if (menu_type == XmMENU_PULLDOWN) {
        str = XmStringCreateSimple(menu_title);
        cascade = XtVaCreateManagedWidget(menu_title,
            xmCascadeButtonGadgetClass, parent,
            XmNsubMenuId,   menu,
            XmNlabelString, str,
            XmNmnemonic,    menu_mnemonic,
            NULL);
        XmStringFree(str);
    } else if (menu_type == XmMENU_OPTION) {
        /* Option menus are a special case, but not hard to handle */
        Arg args[2];
        str = XmStringCreateSimple(menu_title);
        XtSetArg(args[0], XmNsubMenuId, menu);
        XtSetArg(args[1], XmNlabelString, str);
        /* This really isn't a cascade, but this is the widget handle
         * we're going to return at the end of the function.
         */
        cascade = XmCreateOptionMenu(parent, menu_title, args, 2);
        XmStringFree(str);
    }

    /* Now add the menu items */
```

```
    for (i = 0; items[i].label != NULL; i++) {
        /* If subitems exist, create the pull-right menu by calling this
         * function recursively.  Since the function returns a cascade
         * button, the widget returned is used..
         */
        if (items[i].subitems)
            if (menu_type == XmMENU_OPTION) {
                XtWarning("You can't have submenus from option menu items.");
                continue;
            } else
                widget = BuildMenu(menu, XmMENU_PULLDOWN,
                    items[i].label, items[i].mnemonic,
                    items[i].subitems);
        else
            widget = XtVaCreateManagedWidget(items[i].label,
                *items[i].widget_class, menu,
                NULL);

/* Set sensitivity of widget.
 */
         if (items[i].sensitive != True)
           XtSetSensitive(widget, False);

         if (items[i].act_display == True)
            XtVaSetValues(menu,
                XmNmenuHistory, widget,
                NULL);


        /* Whether the item is a real item or a cascade button with a
         * menu, it can still have a mnemonic.
         */
        if (items[i].mnemonic)
            XtVaSetValues(widget, XmNmnemonic, items[i].mnemonic, NULL);

        /* any item can have an accelerator, except cascade menus. But,
         * we don't worry about that; we know better in our declarations.
         */
        if (items[i].accelerator) {
            str = XmStringCreateSimple(items[i].accel_text);
            XtVaSetValues(widget,
                XmNaccelerator, items[i].accelerator,
                XmNacceleratorText, str,
                NULL);
            XmStringFree(str);
        }

        if (items[i].callback)
            XtAddCallback(widget,
                (items[i].widget_class == &xmToggleButtonWidgetClass ||
                 items[i].widget_class == &xmToggleButtonGadgetClass)?
                    XmNvalueChangedCallback : /* ToggleButton class */
                    XmNactivateCallback,      /* PushButton class */
                items[i].callback, items[i].callback_data);
    }

    /* for popup menus, just return the menu; pulldown menus, return
     * the cascade button; option menus, return the thing returned
     * from XmCreateOptionMenu().  This isn't a menu, or a cascade button!
     */
    return menu_type == XmMENU_POPUP? menu : cascade;
};

#ifdef __cplusplus
}  /* Close scope of 'extern "C"' declaration which encloses file. */
#endif
```

```
/* ************************************************************

Name:          font_table.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 12 Sep 92
Remarks:   Specification for the FontTable object.

           The FontTable contains all the necessary information
         about the fonts used in the graph editor, allowing
         the program to refer to them by a font id#.

           It initializes with six predefined font names,
         although more could easily be added.

************************************************************ */
#ifndef FONT_TABLE_H
#define FONT_TABLE_H 1

#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <Xm/MessageB.h>
#include <string.h>
#include "ge_defs.h"

struct _font_record {
  char *_name_ptr;
  XFontStruct  *_font_ptr;
  int _height;
};

class FontTable {
private:
  struct _font_record _font_table[MAXFONTS + 1];
  static Widget _error_tgt;

public:
  static void set_error_tgt(Widget widget) {_error_tgt = widget;}

  FontTable();
  ~FontTable();
  void init(Display *display_ptr);
  int width(int font_id, char *in_string);
  int height(int font_id);
  Font font_id(int font_id);
  char *font_name(int font_id);
};

#endif
```

```
/* ************************************************************

Name:
Author:        Capt Robert M. Dixon
Program:
Date Modified:
Remarks:   Implementation of the FontTable object.

          The FontTable contains all the necessary information
         about the fonts used in the graph editor, allowing
         the program to refer to them by a font id#.

          It initializes with six predefined font names,
         although more could easily be added.


Credits:    Portions of code are adapted from the following:

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.


************************************************************ */
#include <stdio.h>
#include "font_table.h"
#include "ge_utilities.h"
#include "warning.h"

//   Initializes static class variable.

Widget FontTable::_error_tgt = NULL;


FontTable::FontTable() {
  int i;

  for(i = 1; i <= MAXFONTS; i++) {
    _font_table[i]._name_ptr = NULL;
    _font_table[i]._font_ptr = NULL;
  }
}


FontTable::~FontTable() {
  int i;

#ifdef GE_DEBUG
    printf("FontTable Destructor:  array of _name_ptr and _font_ptr\n");
#endif


  for(i = 1; i <= MAXFONTS; i++) {
    //    delete _font_table[i]._name_ptr;
    free(_font_table[i]._name_ptr);
    //    delete _font_table[i]._font_ptr;
    free((char*)_font_table[i]._font_ptr);
  }
}

//   Initializes the font table using fonts defined for the
// given display.

void FontTable::init(Display *display_ptr) {

  _font_table[0]._name_ptr =
                          dup_str("variable");
  _font_table[COURIERBOLD10]._name_ptr =
                          dup_str("*adobe-courier-bold-r*100*");
  _font_table[COURIERBOLD12]._name_ptr =
                          dup_str("*adobe-courier-bold-r*120*");
  _font_table[COURIERBOLD14]._name_ptr =
                          dup_str("*adobe-courier-bold-r*140*");
  _font_table[COURIERMED10]._name_ptr =
                          dup_str("*adobe-courier-medium-r*100*");
  _font_table[COURIERMED12]._name_ptr =
                          dup_str("*adobe-courier-medium-r*120*");
  _font_table[COURIERMED14]._name_ptr =
                          dup_str("*adobe-courier-medium-r*140*");

  _font_table[0]._font_ptr =
    XLoadQueryFont(display_ptr, _font_table[0]._name_ptr);
  _font_table[COURIERBOLD10]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERBOLD10]._name_ptr);
  _font_table[COURIERBOLD12]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERBOLD12]._name_ptr);
  _font_table[COURIERBOLD14]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERBOLD14]._name_ptr);
  _font_table[COURIERMED10]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERMED10]._name_ptr);
  _font_table[COURIERMED12]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERMED12]._name_ptr);
  _font_table[COURIERMED14]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERMED14]._name_ptr);
```

```
  _font_table[0]._height =
    _font_table[0]._font_ptr->ascent +
      _font_table[0]._font_ptr->descent;

  _font_table[COURIERBOLD10]._height =
    _font_table[COURIERBOLD10]._font_ptr->ascent +
      _font_table[COURIERBOLD10]._font_ptr->descent;

  _font_table[COURIERBOLD12]._height =
    _font_table[COURIERBOLD12]._font_ptr->ascent +
      _font_table[COURIERBOLD12]._font_ptr->descent;

  _font_table[COURIERBOLD14]._height =
    _font_table[COURIERBOLD14]._font_ptr->ascent +
      _font_table[COURIERBOLD14]._font_ptr->descent;

  _font_table[COURIERMED10]._height =
    _font_table[COURIERMED10]._font_ptr->ascent +
      _font_table[COURIERMED10]._font_ptr->descent;

  _font_table[COURIERMED12]._height =
    _font_table[COURIERMED12]._font_ptr->ascent +
      _font_table[COURIERMED12]._font_ptr->descent;

  _font_table[COURIERMED14]._height =
    _font_table[COURIERMED14]._font_ptr->ascent +
      _font_table[COURIERMED14]._font_ptr->descent;
}

//   Returns the width of the given string in the given font
// in pixels.

int FontTable::width(int font_id, char *in_string) {

  if(in_string == NULL)
    return 0;
  else
    if(font_id > MAXFONTS)
      warning(_error_tgt,"Font Table entered with font too big.");
    else
      return XTextWidth(_font_table[font_id]._font_ptr,
                        in_string, strlen(in_string));
}

//   Returns the height of the given font in pixels.

int  FontTable::height(int font_id) {

  if(font_id <= MAXFONTS)
    return _font_table[font_id]._height;
  else {
    warning(_error_tgt,"Font Table entered with font too big.");
    return 0;
  }
}

//   Returns font information needed by some X functions.

Font FontTable::font_id(int font_id) {

  if(font_id <= MAXFONTS)
    return _font_table[font_id]._font_ptr->fid;
  else {
    warning(_error_tgt,"Font Table entered with font too big.");
    return 0;
  }
}

//   Returns the name of the given font.

char *FontTable::font_name(int font_id) {

  if(font_id <= MAXFONTS)
    return _font_table[font_id]._name_ptr;
  else {
    warning(_error_tgt,"Font Table entered with font too big.");
    return NULL;
  }
}
```

```
/* ************************************************************

Name:          ge_defs.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 12 Sep 92
Remarks:   Provides the common type definitions and defines
         for the modules in the graph_editor program.


************************************************************ */
#ifndef ge_defs_h
#define ge_defs_h 1

#include "ge_interface.h"     /* BOOLEAN, TRUE and FALSE defined */

enum CLASS_DEF{GRAPHOBJECT, OPERATOROBJECT, STREAMOBJECT,
               GRAPHOBJECTLIST, SPLINEOBJECT};

enum TOOL_STATE{OPERATOR_TOOL, TERMINATOR_TOOL, STREAM_TOOL, SELECT_TOOL,
                TYPES_TOOL, SPEC_TOOL, TIMERS_TOOL, INFORMAL_TOOL};

enum DRAW_STYLE{SOLID, DOTTED, ERASE};

typedef int GE_STATUS;
typedef int EXTERN_STATUS;
typedef int COLOR;

typedef struct xypair {
  int x, y;
} XYPAIR;
```

```
#define ENDED 2
#define SUCCEEDED 1
#define FAILED 0
#define INPUT_LINE_SIZE 100
#define CIRCLE_BEGIN 0
#define FULL_CIRCLE 360 * 64
#define HANDLESIZE 5

/*   Return codes used to tell the syntax-directed editor
  what to do with the incoming data, and whether any
  level changes are required.
*/

#define UP -1
#define SAME 0
#define ERROR -3
#define NOUPDATE -4

#define NO_EXTERNAL 0
#define FROM_EXTERNAL 1
#define TO_EXTERNAL 2

#define HITFUDGE 5

#define NULL_VALUE 0
#define MAXDELETEDOPS 100
#define MAXTEXTLINES 100

#include "resources.h"
#endif
```

```
/* ************************************************************

Name:     ge_driver.C
Author:   Ken Moeller
Program:  graph_editor
Remarks:


History:
  01   96/09/29  Ken Moeller

************************************************************ */
#include <unistd.h>

#include "inter_process_utilities.h"
#include "graph_editor.h"
#include "get_unique_id.h"

extern  XferBuffer *xfer;
int    Global_argc;
char **Global_argv;

main(int argc, char *argv[])
{
  int
        ge_in,
ge_out;

  int  chkWord;

  ACTION_NODE     next_action;
  ACTION          action = &next_action;

  GRAPH_DESC_NODE graph;
  GRAPH_DESC      current_graph = &graph;

  ERROR_MSGS      sde_error_msgs = NULL;

  Global_argc = argc;
  Global_argv = argv;

  xfer = NULL;
  create_xfer_buf(&xfer);

  init_action(action);
  init_graph_desc(current_graph);

  sscanf(argv[1], "%d", &ge_in);
  sscanf(argv[2], "%d", &ge_out);

  init_id_server();

  next_action.reinvoke = true;


  readChkWord(&chkWord, ge_in);
  writeChkWord(chkWord, ge_out);

  if (chkWord == CHKWORD) {
    /* Loop until no longer reinvoked. */
    while (next_action.reinvoke) {

      /* read data from sde */
      readGraphDesc(current_graph, xfer, ge_in);
      readErrorMsgs(&sde_error_msgs, xfer, ge_in);

      edit_graph(current_graph, action, sde_error_msgs);

      /* write data back to sde */
      writeGraphDesc(current_graph, xfer, ge_out);
      writeAction(action, xfer, ge_out);
    }

#ifdef GE_DEBUG
    printf("\nGE Xfer Buffer size: %d\n", xfer->Max);
#endif

  }
  else {
    printf("Error communicating with SDE\n");
  }

  close(ge_in);
  close(ge_out);

  sleep(10000); /* Give SDE time to kill process                    */
                /* Needed so that the pipe is not killed before the */
                /* SDE has time to read it.                         */

  return 0;
}
```

```
/* ************************************************************

Name:          ge_interface.h

************************************************************ */
#ifndef GE_INTERFACE_H
#define GE_INTERFACE_H 1

typedef unsigned short BOOLEAN;
/*
#include "/local/syn/src/include/lang.h"
*/

#define US 0               /* microsec */
#define MS 1               /* millisec */
#define SECOND 2           /* sec */
#define MINUTE 3           /* min */
#define HOUR   4           /* hours */

#define UNDEFINED_OPNUM 0  /* constant representing undefined op num */
#define EXTERNAL_VERTEX_NUM 0 /* OP_ID for an external stream */
#define UNDEFINED_TIME -1  /* constant representing undefined time value */

#define UNPROTECTED  0     /* no data trigger   */
#define BY_SOME      1     /* triggered by some */
#define BY_ALL       2     /* triggered by all  */

#define NTC          0     /* non time critical */
#define PERIODIC     1     /* periodic          */
#define SPORADIC     2     /* sporadic          */

#define BUFFER_SIZE  1000  /* use by routines in sde.string_unit.foreign.ssl */
                           /* to hold the text from from sde text capture    */

#define UPDATE_TREE   0    /* only update internal structure */
#define CHECK_SYNTAX  1    /* update sde syntax tree */
#define SAVE_TO_DISK  2    /* update syntax tree and save to file */
#define REVERT        3    /* delete syntax tree and rebuild it from file */
#define ABANDON       4    /* abandon work in current level */

#define true  (1)
#define false (0)

#define DEFAULT_IMPL_LANG "Ada"

typedef int GE_ACTION_TYPE;   /* enumeration type
                                 (UPDATE_TREE, CHECK_SYNTAX,
                                  SAVE_TO_DISK, REVERT, ABANDON) */

typedef unsigned int OP_ID;       /* unique identifier numbers for
                                     OPERATORs, 1 .. # ops          */
typedef unsigned int ST_ID;       /* unique identifier numbers for
                                     streams, 1 .. # streams        */

/************************************************************ */
/* typdef for an ID linked list                               */
/************************************************************ */
typedef struct id_list {
  char* id;
  struct id_list* next;
}ID_NODE, *ID_LIST;


/************************************************************ */
/* typedef for an OPERATOR                                    */
/************************************************************ */
typedef struct op_str {
  OP_ID id;                    /* cannot be changed after creation
                                  this is equal to the vertex_num,
                                  assigned by ge and used by sde  */

  OP_ID
op_num;                /* only assign and used by sde     */

  /* info about location of operator icon */
  int
        x,                     /* x_position of the circle's center point */
        y,                     /* y_position of the circle's center point */
        radius,
        color;

  /* info about the label */
  char *label;
  int
        label_font,
        label_x_offset,        /* actual x_position = x + label_x_offset */
        label_y_offset;        /* actual y_position = y + label_y_offset */

  /* info about timing constraints */
  int
        timing_type;           /* NTC, PERIOD, SPORADIC      */

  /* info about MET */
  int
        met,                   /* UNDEFINED_TIME if no met */
        met_unit,              /* US, MS, SECOND, MINUTE, HOUR */
        met_font,
        met_x_offset,          /* actual x_position = x + met_x_offset */
        met_y_offset;          /* actual y_position = y + met_y_offset */
  ID_LIST
        met_reqmts;            /* requirements trace           */

  /* info about PERIOD */
  int
```

```
      period,                 /* UNDEFINED_TIME if no period */
      period_unit;            /* US, MS, SECOND, MINUTE, HOUR */
ID_LIST
      period_reqmts;          /* requirements trace           */

/* info about FINISH WITHIN */
int
      fw,                     /* UNDEFINED_TIME if no finish within */
      fw_unit;                /* US, MS, SECOND, MINUTE, HOUR */
ID_LIST
      fw_reqmts;              /* requirements trace           */

/* info about MAXIMUM RESPONSE TIME */
int
      mrt,                    /* UNDEFINED_TIME if no maximum response time */
      mrt_unit;               /* US, MS, SECOND, MINUTE, HOUR */
ID_LIST
      mrt_reqmts;             /* requirements trace           */

/* info about MINIMUM CALLING PERIOD */
int
      mcp,                    /* UNDEFINED_TIME if no minimum calling period*/
      mcp_unit;               /* US, MS, SECOND, MINUTE, HOUR */
ID_LIST
      mcp_reqmts;             /* requirements trace           */

/* info about triggering */
int
      trigger_type;           /* UNPROTECTED, BY_SOME, BY_ALL */
ID_LIST
      trigger_set;
char*
      if_condition;           /* triggered_if condition, set to true
                                 if not specified,
                                 parameter to be forwarded to mini-sde
                                 EDIT_IF_COND(char* if_condition)     */
ID_LIST
      trigger_reqmts;         /* requirements trace           */

/* info about output guard */
char*
      output_guard_list;      /* parameter to be forwarded to mini-sde
                                 EDIT_OUTPUT_GUARD(char* out_guard_list) */

/* info about exception */
char*
      exception_list;         /* parameter to be forwarded to mini-sde
                                 EDIT_EXCEPTION(char* exception_list) */

/* info about timer op */
char*
      timer_op_list;          /* parameter to be forwarded to mini-sde
                                 EDIT_TIMER_OP(char* timer_op_list) */

  /* info about key_words for operator spec */
  ID_LIST
        key_word_list;        /* list of keywords f*/

  /* info about informal description operator spec */
  char*
        operator_informal_desc;

  /* info about formal description operator spec */
  char*
        operator_formal_desc;

  /* info about the implementation of the operator */
  char*
        operator_impl_lang;

  /* operator properties */
  BOOLEAN
        is_composite,
        is_terminator;

  /* info for house keeping */
  BOOLEAN
        is_new,       /* is_new is set by the GE and
cleared by the SDE' New_Operator procedure */
        is_modified,
        is_deleted;

}OP_NODE, *OPERATOR;


/************************************************************ */
/* typedef for a linked list of OPERATOR                      */
/************************************************************ */
typedef struct op_list_node {
  OPERATOR op;
  struct op_list_node *next;
}OP_LIST_NODE, *OP_LIST;


/************************************************************ */
/* typedef for SPLINE linked list                             */
/************************************************************ */
typedef struct spline_node {
  int
        x,              /* each (x,y) represent an interval control point */
        y;              /* for the spline, not including end points.      */
  struct spline_node*
        next;
}SPLINE_NODE, *SPLINE_PTR;


/************************************************************ */
```

```
/* typedef for a STREAM                                          */
/************************************************************* */
typedef struct st_str{
  ST_ID id;          /* cannot be changed after creation */

  /* info about label */
  char *label;
  int
        label_font,
        label_x_offset,  /* actual x_position = x_mid_point + label_x_offset */
        label_y_offset;  /* actual Y_position = y_mid_point + label_y_offset */
                         /* x_mid_point = 0.5 * ( from->x + to->x)           */
                         /* y_mid_point = 0.5 * ( from->y + to->y)           */

  /* the two endpoints of the stream */
  OP_ID
        from,
        to;

  /* linked list of SPLINE_NODE defining the shape of the edge on the display*/
  SPLINE_PTR
        arc;               /* null pointer if the arc is a
                              straight line */


  /* info about LATENCY */
  int
        latency,                   /* UNDEFINED_TIME if not specified */
        latency_unit,              /* US, MS, SECOND, MINUTE, HOUR */
        latency_font,
        latency_x_offset,  /* actual x_position=x_mid_point+latency_x_offset*/
        latency_y_offset;  /* actual Y_position=y_mid_point+latency_y_offset*/

  /* stream type */
  char*
        stream_type_name;

  /* initial value if it is a state stream */
  char*
        state_initial_value;

  /* stream visible properties */
  BOOLEAN
        is_state_variable;

  /* info for house keeping */
  BOOLEAN
        is_new,            /* is_new is set by the GE and
                              cleared by the SDE' Make_Edge_List
     function */
        is_modified,
        is_deleted;
}ST_NODE, *STREAM;


/************************************************************* */
/*typedef for a linked list of STREAM                          */
/************************************************************* */
typedef struct st_list_node {
  STREAM st;
  struct st_list_node *next;
}ST_LIST_NODE, *ST_LIST;


/************************************************************* */
/*typedef for operator specification interface attributes       */
/************************************************************* */
typedef struct attributes_list {
  char*   attr;
  ID_LIST reqmts;
  struct attributes_list *next;
}AT_NODE, *AT_LIST;


/************************************************************* */
/* typedef for the graph description structure                  */
/************************************************************* */
typedef struct graph_desc_node{
  /* From SDE to GE */
  char* root_op_name;          /* name of the root operator          */
  int   root_op_num;           /* unique op_num of the root operator */
  char* parent_op_name;        /* name of the parent of the current operator,
                                  the parent of the root operator is  */
  int   parent_op_num;         /* unique op_num of the parent operator */
  char* current_op_name;       /* name of the current operator
                                  whose dataflow graph is being edited  */
  int   current_op_num;        /* unique op_num of the current operator */


  /* From SDE to GE */
  ST_LIST input_list;          /* list of input streams                */
  ST_LIST output_list;         /* list of output streams               */
                          /* NOTE: only label, label_font, stream_type_name,
                             state_initial_value, is_state_variable are
                             meaningful in the input_list and output_list */

  /* From SDE to GE */
  int     cur_op_spec_met;      /* MET is kept separate from the spec     */
  int     cur_op_spec_met_unit; /* interface.  Still, only the reqmts can */
  /* MTS 11/25/96
     change cur_op_is_terminator from int to BOOLEAN */
  BOOLEAN     cur_op_is_terminator;

  /* Bi-directional between SDE and GE */
  char*   cur_op_spec;          /* Specification of current operator which */
```

```
                                /* is edited with mini-sde.                */

  /* Bi-directional between SDE and GE */
  OP_LIST operator_list;
  ST_LIST stream_list;
  ID_LIST timer_list;
  char*   graph_informal_desc;

  /* From SDE to GE */
  ID_LIST avail_impl_langs;     /* An ID_LIST of available languages from */
                                /* which an operator can be implemented   */

  /* Bi-directional between SDE and GE */
  char*   global_types;         /* SDE output of all types                */

}GRAPH_DESC_NODE, *GRAPH_DESC;


/************************************************************ */
/* typedef for the ACTION type                                */
/************************************************************ */
typedef struct action_node {
  GE_ACTION_TYPE option;        /* UPDATE_TREE, CHECK_SYNTAX, SAVE_TO_DISK,
                                   REVERT, ABANDON       */
  BOOLEAN        reinvoke;      /* true if need to reopen GE              */
  char*          next_op;       /* name of the next operator to be edited,
                                   only meaningful if reinvoke == true    */
  int            next_op_num;   /* set to UNDEFINED_OPNUM if next_op is
                                   a newly created bubble                 */
} ACTION_NODE, *ACTION;


/************************************************************ */
/* typedef for the ERROR_MSGS type                            */
/************************************************************ */
typedef struct error_node {
  OP_ID  parent_op_num;         /* Parent is the operator above the */
  char*  parent_op_label;       /* operator with the error          */
  OP_ID  cur_op_num;
  char*  cur_op_label;
  char*  msg;                   /* Text of error message            */
  struct error_node *next;
} ERROR_NODE, *ERROR_MSGS;


#endif
```

```
/* ***********************************************************

Name:          ge_interface_labels.h

History:

  @1   96/10/07  Ken Moeller
       Changes to ge_interface.h

*********************************************************** */
#ifndef GE_INTERFACE_LABELS_H
#define GE_INTERFACE_LABELS_H 1

static char *time_units[] = {
  "Undefined_Time",
  "US",
  "MS",
  "SECOND",
  "MINUTE",
  "HOUR" };

static char *trigger_conds[] = {
  "UNPROTECTED",
  "BY_SOME",
  "BY_ALL" };

static char *timing_types[] = {
  "NTC",
  "PERIODIC",
  "SPORADIC" };

static char *lb_arc                    = "arc";
static char *lb_color                  = "color";
static char *lb_exception_list         = "exception_list";
static char *lb_from                   = "from";
static char *lb_fw                     = "fw";
static char *lb_fw_reqmts              = "fw_reqmts";
static char *lb_fw_unit                = "fw_unit";
static char *lb_id                     = "id";
static char *lb_if_condition           = "if_condition";
static char *lb_is_composite           = "is_composite";
static char *lb_is_deleted             = "is_deleted";
static char *lb_is_modified            = "is_modified";
static char *lb_is_new                 = "is_new";
static char *lb_is_state_variable      = "is_state_variable";
static char *lb_is_terminator          = "is_terminator";
static char *lb_key_word_list          = "key_word_list";
static char *lb_label                  = "label";
static char *lb_label_font             = "label_font";
static char *lb_label_x_offset         = "label_x_offset";
static char *lb_label_y_offset         = "label_y_offset";
static char *lb_latency                = "latency";
static char *lb_latency_font           = "latency_font";
static char *lb_latency_unit           = "latency_unit";
static char *lb_latency_x_offset       = "latency_x_offset";
static char *lb_latency_y_offset       = "latency_y_offset";
static char *lb_mcp                    = "mcp";
static char *lb_mcp_reqmts             = "mcp_reqmts";
static char *lb_mcp_unit               = "mcp_unit";
static char *lb_met                    = "met";
static char *lb_met_font               = "met_font";
static char *lb_met_reqmts             = "met_reqmts";
static char *lb_met_unit               = "met_unit";
static char *lb_met_x_offset           = "met_x_offset";
static char *lb_met_y_offset           = "met_y_offset";
static char *lb_mrt                    = "mrt";
static char *lb_mrt_reqmts             = "mrt_reqmts";
static char *lb_mrt_unit               = "mrt_unit";
static char *lb_op_num                 = "op_num";                 /* @1 */
static char *lb_operator_formal_desc   = "operator_formal_desc";
static char *lb_operator_impl_lang     = "operator_impl_lang";
static char *lb_operator_informal_desc = "operator_informal_desc";
static char *lb_output_guard_list      = "output_guard_list";
static char *lb_period                 = "period";
static char *lb_period_reqmts          = "period_reqmts";
static char *lb_period_unit            = "period_unit";
static char *lb_radius                 = "radius";
static char *lb_state_initial_value    = "state_initial_value";
static char *lb_stream_type_name       = "stream_type_name";
static char *lb_timer_op_list          = "timer_op_list";
static char *lb_timing_type            = "timing_type";
static char *lb_to                     = "to";
static char *lb_trigger_reqmts         = "trigger_reqmts";
static char *lb_trigger_set            = "trigger_set";            /* @1 */
static char *lb_trigger_type           = "trigger_type";
static char *lb_x                      = "x";
static char *lb_y                      = "y";


#endif
```

```
/* ************************************************************

Name:          ge_utilities.h
Author:        Lange
Program:       graph_editor
Date Modified: 5 Oct 96

This is basically a slightly more object
oriented view of the ge_interface structures

************************************************************
History:

  @1   96/10/05  Ken Moeller
       Several items were not being released, but set to NULL
       instead.

  @2   96/10/05  Ken Moeller
       Added a routine for making a string which contains
       the time and units.

  @3   96/10/06  Ken Moeller
       New routine for returning an OPERATOR pointer for a
       given Operator id.

  @4   96/10/18  Ken Moeller
       New routine for validating a PSDL integer_literal

  @5   96/10/18  Ken Moeller
       New routine for testing text to see if it is all white
       space.

  @6   96/11/25  M.T. Shing
       New routine for copying, replacing and releasing of AT_LIST

  @7   96/11/25  Ken Moeller
       New routine for validating a PSDL id

  @8   96/11/25  Ken Moeller
       New routine for determining if an id is a PSDL keyword

  @9   96/11/27  M.T. Shing
       Add routine to search AT_LIST

  @10  96/11/27  Ken Moeller
       New routine to validate a PSDL op_id

  @11  96/11/27  Ken Moeller
       New routine to pull out a PSDL id, return true if valid

  @12  96/11/27  Ken Moeller
       New routine to validate a PSDL type_name

  @13  96/12/2   M.T.Shing
       Add routines to copy OP_LIST and ST_LIST.

************************************************************ */
#ifndef GE_UTILITIES_H
#define GE_UTILITIES_H 1

#include <string.h>
#include "ge_interface.h"

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

ID_LIST id_list_copy();
void    id_list_release();
void id_list_replace();

void op_release();
OPERATOR op_copy();                                    /* @13 */

void op_list_release();
OP_LIST op_list_copy();                                /* @13 */

SPLINE_PTR arc_copy();
void arc_release();
void st_release();
STREAM st_copy();                                /* @13 */

void st_list_release();
ST_LIST st_list_copy();                                /* @13 */

void graph_release();

void init_action();
void init_graph_desc();
void init_operator();
void init_stream();

void err_msgs_release();

char*   time_with_units();
OPERATOR operator_id_ptr();
char* dup_str();

BOOLEAN valid_num_string();
BOOLEAN valid_integer_literal();
BOOLEAN valid_id();                                    /* @7 */
BOOLEAN parse_id();                                    /* @11 */
BOOLEAN valid_op_id();                                 /* @10 */
BOOLEAN valid_type_name();                             /* @12 */
```

```
BOOLEAN is_keyword();
BOOLEAN white_space();                                  /* @8 */

AT_LIST at_list_copy();                                 /* @6 */
void    at_list_release();                              /* @6 */
void at_list_replace();                                /* @6 */

#else

ID_LIST id_list_copy(ID_LIST y);
void    id_list_release(ID_LIST X);
void id_list_replace(ID_LIST *x, ID_LIST y);

void op_release(OPERATOR X);
OPERATOR op_copy(OPERATOR X);                           /* @13 */

void op_list_release(OP_LIST X);
OP_LIST op_list_copy(OP_LIST X);                        /* @13 */

SPLINE_PTR arc_copy(SPLINE_PTR y);
void arc_release(SPLINE_PTR X);
void st_release(STREAM X);
STREAM st_copy(STREAM X);                              /* @13 */

void st_list_release(ST_LIST X);
ST_LIST st_list_copy(ST_LIST X);                        /* @13 */

void graph_release(GRAPH_DESC x);

void init_action(ACTION act);
void init_graph_desc(GRAPH_DESC gdPtr);
void init_operator(OPERATOR OpPtr);
void init_stream(STREAM StPtr);

void err_msgs_release(ERROR_MSGS *ErrPtr);

char*   time_with_units(int time, int unit);            /* @2 */
OPERATOR operator_id_ptr(OP_LIST ptr, int id);          /* @3 */
char* dup_str(char* ptr);

BOOLEAN valid_num_string(char *num);
BOOLEAN valid_integer_literal(char *num, int *value);  /* @4 */
BOOLEAN valid_id(char *id);                             /* @7 */
BOOLEAN parse_id(char *ID, int *idLen,                  /* @11 */
 BOOLEAN allow_types);
BOOLEAN valid_op_id(char *opID);                        /* @10 */
BOOLEAN valid_type_name(char *typeName);                /* @12 */
BOOLEAN is_keyword(char *id, BOOLEAN allow_types);      /* @8 */
BOOLEAN white_space(char *text);                        /* @5 */

AT_LIST at_list_copy(AT_LIST y);                        /* @6 */
void    at_list_release(AT_LIST X);                     /* @6 */
void at_list_replace(AT_LIST *x, AT_LIST y);           /* @6 */
AT_LIST at_list_search(char *label, AT_LIST y);         /* @9 */
#endif

#ifdef __cplusplus
}
#endif

#endif
```

```
/* ************************************************************

Name:          ge_utilities.c
Author:        Lange
Program:       graph_editor
Date Modified: 5 Oct 96

************************************************************
History:

  @1   96/10/05  Ken Moeller
       Several items were not being released, but set to NULL
       instead.

  @2   96/10/05  Ken Moeller
       Added a routine for making a string which contains
       the time and units.

  @3   96/10/06  Ken Moeller
       New routine for returning an OPERATOR pointer for a
       given Operator id.

  @4   96/10/07  Ken Moeller
       Moved utility.C and utility.h to ge_utilities in order
       to group all ge_interface routines in one area.

  @5   96/10/07  Ken Moeller
       Changed dynamic memory over from C++ to C versions.

  @6   96/10/07  Ken Moeller
       Changes made to support changes in ge_interface.h.

  @7   96/10/07  Ken Moeller
       New routines to handle streams and splines.

  @8   96/10/18  Ken Moeller
       New routine to validate a PSDL integer_literal

  @9   96/10/18  Ken Moeller
       New routine to check if text consists only of white space.

  @10  96/11/25  M.T. Shing
       New routine for copying, replacing and releasing of AT_LIST

  @11  96/11/25  Ken Moeller
       New routine to validate a PSDL id

  @12  96/11/27  M.T. Shing
       Add routine to search AT_LIST

  @13  96/11/27  Ken Moeller
       New routine to validate a PSDL op_id

  @14  96/11/27  Ken Moeller
       New routine to pull out a PSDL id, return true if valid

  @15  96/11/27  Ken Moeller
       New routine to validate a PSDL type_name

  @16  96/12/2   M.T.Shing
       Add routines to copy OP_LIST and ST_LIST.

************************************************************ */

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include "ge_utilities.h"

#ifdef __cplusplus
extern "C" {
#endif

/*****************************************************************************
 * id_list_copy(ID_LIST y)                                                   *
 *                                                                           *
 * Written by Doug Lange 9/8/96.  Builds and returns an ID_LIST              *
 * Deep copy of an ID_LIST.  The new ID_LIST may be modified in              *
 * that any ID_NODE with an ID of NULL will be returned with an              *
 * ID containing an empty string ("").                                       *
 *****************************************************************************/
#ifdef _NO_PROTO
ID_LIST id_list_copy(y)
ID_LIST y;
{
#else
ID_LIST id_list_copy(ID_LIST y) {
#endif
  /* y is the original ID_LIST,
     x is the new ID_LIST,
     temp1 is used to build the new ID_LIST,
     temp2 is used to step through the original ID_LIST.        */
  ID_LIST temp1, temp2;

  if (y != NULL) {      /* build the new list */
    ID_LIST x = (ID_LIST) malloc(sizeof(ID_NODE));        /* @5 */
    x->next = NULL;
    temp1 = x;
    temp2 = y;
    while (temp2 != NULL) {
      temp1->id = dup_str(temp2->id);

      if (temp2->next != NULL) {
temp1->next = (ID_LIST) malloc(sizeof(ID_NODE));  /* @5 */
temp1 = temp1->next;
      }
```

```
      temp1->next = NULL;
      temp2 = temp2->next;
    }
    return(x);
  }
  else
    return(NULL);
}


/*************************************************************************
 * id_list_release(ID_LIST X)                                            *
 *                                                                       *
 * Written by Doug Lange 9/8/96.  Releases cells.                        *
 * Recovers all dynamic memory allocated to an ID_LIST.                  *
 *************************************************************************/
#ifdef _NO_PROTO
void id_list_release(X)
ID_LIST X;
{
#else
void id_list_release(ID_LIST X) {
#endif
  ID_LIST  temp1;

  while (X != NULL) {     /* delete x's cells */
    temp1 = X->next;
    free(X->id);            /* @5 */
    free((char *)X);        /* @5 */
    X = temp1;              /* will return NULL on exit */
  }
  return;
}


/*************************************************************************
 * id_list_replace(ID_LIST *x, ID_LIST y)                                *
 *                                                                       *
 * Written by Doug Lange 9/8/96.  Replaces what was in x with y while    *
 * ensuring releasing cells.                                             *
 * This routine performs a deep copy replacement of one ID_LIST with     *
 * another ID_LIST.                                                      *
 *************************************************************************/
#ifdef _NO_PROTO
void id_list_replace(x, y)
ID_LIST *x;
ID_LIST y;
{
#else
void id_list_replace(ID_LIST *x, ID_LIST y) {
#endif

  if (x != NULL) {
    id_list_release(*x);
    *x = id_list_copy(y);
  }
  else {
#ifdef GE_DEBUG
  printf("ERROR:  id_list_replace to a NULL location.\n");
#endif
  }
}


/*************************************************************************
 * op_release(OPERATOR X)                                                *
 *                                                                       *
 * Manages operators and their lists Doug Lange 9/11/96                  *
 * Recovers all dynamic memory contained within the OP_NODE.  OP_NODE    *
 * memory itself is not recovered.                                       *
 * @5 changed over to C dynamic memory library functions.                *
 *************************************************************************/
#ifdef _NO_PROTO
void op_release(X)
OPERATOR X;
{
#else
void op_release(OPERATOR X) {
#endif

  if (X != NULL) {

    free(X->label);                  X->label                  = NULL;
    free(X->if_condition);           X->if_condition           = NULL;
    free(X->output_guard_list);      X->output_guard_list      = NULL;
    free(X->exception_list);         X->exception_list         = NULL;
    free(X->timer_op_list);          X->timer_op_list          = NULL;
    free(X->operator_informal_desc); X->operator_informal_desc = NULL;
    free(X->operator_formal_desc);   X->operator_formal_desc   = NULL;
    free(X->operator_impl_lang);     X->operator_impl_lang     = NULL;

    id_list_release(X->met_reqmts);
    X->met_reqmts = NULL;
    id_list_release(X->period_reqmts);
    X->period_reqmts = NULL;
    id_list_release(X->fw_reqmts);                /* @1 was commented out */
    X->fw_reqmts = NULL;
    id_list_release(X->mrt_reqmts);
    X->mrt_reqmts = NULL;
    id_list_release(X->mcp_reqmts);
    X->mcp_reqmts = NULL;
    id_list_release(X->trigger_set);              /* @6 */
    X->trigger_set = NULL;
    id_list_release(X->trigger_reqmts);           /* @1 was commented out */
    X->trigger_reqmts = NULL;
    id_list_release(X->key_word_list);
```

```
    X->key_word_list = NULL;
  }
}


/*****************************************************************************
 * op_copy(OPERATOR X)                                                  @16 *
 *                                                                          *
 * Deep copy of an Operator                                                 *
 *****************************************************************************/
#ifdef _NO_PROTO
OPERATOR op_copy(X)
OPERATOR X;
{
#else
OPERATOR op_copy(OPERATOR X) {
#endif

  OPERATOR temp_op;

  if (X != NULL) {
    temp_op = (OPERATOR) malloc(sizeof(OP_NODE));

    temp_op->id = X->id;
    temp_op->op_num = X->op_num;
    temp_op->x = X->x;
    temp_op->y = X->y;
    temp_op->radius = X->radius;
    temp_op->color = X->color;
    temp_op->label =  strdup(X->label);
    temp_op->label_font =  X->label_font;
    temp_op->label_x_offset =  X->label_x_offset;
    temp_op->label_y_offset =  X->label_y_offset;
    temp_op->timing_type =  X->timing_type;
    temp_op->met =  X->met;
    temp_op->met_unit =  X->met_unit;
    temp_op->met_font =  X->met_font;
    temp_op->met_x_offset =  X->met_x_offset;
    temp_op->met_y_offset =  X->met_y_offset;
    temp_op->met_reqmts =  id_list_copy(X->met_reqmts);
    temp_op->period =  X->period;
    temp_op->period_unit =  X->period_unit;
    temp_op->period_reqmts =  id_list_copy(X->period_reqmts);
    temp_op->fw =  X->fw;
    temp_op->fw_unit =  X->fw_unit;
    temp_op->fw_reqmts =  id_list_copy(X->fw_reqmts);
    temp_op->mrt =  X->mrt;
    temp_op->mrt_unit =  X->mrt_unit;
    temp_op->mrt_reqmts =  id_list_copy(X->mrt_reqmts);
    temp_op->mcp =  X->mcp;
    temp_op->mcp_unit =  X->mcp_unit;
    temp_op->mcp_reqmts =  id_list_copy(X->mcp_reqmts);
    temp_op->trigger_type =  X->trigger_type;
    temp_op->trigger_set =  id_list_copy(X->trigger_set);
    temp_op->if_condition = strdup(X->if_condition);
    temp_op->trigger_reqmts =  id_list_copy(X->trigger_reqmts);
    temp_op->output_guard_list = strdup(X->output_guard_list);
    temp_op->exception_list = strdup(X->exception_list);
    temp_op->timer_op_list = strdup(X->timer_op_list);
    temp_op->key_word_list =  id_list_copy(X->key_word_list);
    temp_op->operator_informal_desc = strdup(X->operator_informal_desc);
    temp_op->operator_formal_desc = strdup(X->operator_formal_desc);
    temp_op->operator_impl_lang = strdup(X->operator_impl_lang);
    temp_op->is_composite =  X->is_composite;
    temp_op->is_terminator =  X->is_terminator;
    temp_op->is_new =  X->is_new;
    temp_op->is_modified =  X->is_modified;
    temp_op->is_deleted =  X->is_deleted;

    return(temp_op);
  }
  else
    return(NULL);
}


/*****************************************************************************
 * op_list_release(OP_LIST X)                                               *
 *                                                                          *
 * Recovers all dynamic memory allocated to an OP_LIST.                     *
 *****************************************************************************/
#ifdef _NO_PROTO
void op_list_release(X)
OP_LIST X;
{
#else
void op_list_release(OP_LIST X) {
#endif
  OP_LIST tempOP;

  while (X != NULL) {
    tempOP = X->next;
    op_release(X->op);
    free((char *) X->op);         /* kbm */
    free((char *) X);             /* kbm, also need to free OP_LIST_NODE */
    X = tempOP;
  }
}


/*****************************************************************************
 * op_list_copy(OP_LIST X)                                              @16 *
 *                                                                          *
 * Deep copy of an OP_LIST.                                                 *
 *****************************************************************************/
#ifdef _NO_PROTO
```

```
OP_LIST op_list_copy(X)
OP_LIST X;
{
#else
OP_LIST op_list_copy(OP_LIST X) {
#endif
  OP_LIST tempOP;

  if (X == NULL)
    return(NULL);
  else
  {
    tempOP = (OP_LIST) malloc (sizeof(OP_LIST_NODE));
    tempOP->op = op_copy(X->op);
    tempOP->next = op_list_copy(X->next);

    return(tempOP);
  }
}


/*************************************************************************
 * arc_copy(SPLINE_PTR y)                                          @7 *
 *                                                                    *
 * Deep copy of an SPLINE_PTR.                                        *
 *************************************************************************/
#ifdef _NO_PROTO
SPLINE_PTR arc_copy(y)
SPLINE_PTR y;
{
#else
SPLINE_PTR arc_copy(SPLINE_PTR y) {
#endif
  /* y is the original SPLINE_PTR,
     x is the new SPLINE_PTR,
     temp1 is used to build the new SPLINE_PTR,
     temp2 is used to step through the original SPLINE_PTR.        */
  SPLINE_PTR temp1, temp2;

  if (y != NULL) {
    SPLINE_PTR x = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
    x->next = NULL;
    temp1 = x;
    temp2 = y;
    while (temp2 != NULL) {
      temp1->x = temp2->x;
      temp1->y = temp2->y;

      if (temp2->next != NULL) {
temp1->next = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
temp1 = temp1->next;
      }
      temp1->next = NULL;
      temp2 = temp2->next;
    }
    return(x);
  }
  else
    return(NULL);
}


/*************************************************************************
 * arc_release(SPLINE_PTR X)                                       @7 *
 *                                                                    *
 * Recovers all dynamic memory allocated to a SPLINE_PTR.             *
 *************************************************************************/
#ifdef _NO_PROTO
void arc_release(X)
SPLINE_PTR X;
{
#else
void arc_release(SPLINE_PTR X) {
#endif

  SPLINE_PTR tempSP;

  while (X != NULL) {
    tempSP = X->next;
    free((char *) X);
    X = tempSP;
  }
}


/*************************************************************************
 * st_release(STREAM X)                                            @7 *
 *                                                                    *
 * Recovers all dynamic memory allocated within a STREAM_NOOE.  STREAM_NODE*
 * memory itself is not recovered.                                    *
 *************************************************************************/
#ifdef _NO_PROTO
void st_release(X)
STREAM X;
{
#else
void st_release(STREAM X) {
#endif

  if (X != NULL) {
    free(X->label);                 X->label               = NULL;
    free(X->stream_type_name);      X->stream_type_name    = NULL;
    free(X->state_initial_value);   X->state_initial_value = NULL;

    /* MTS 11/7/96 change NULL to EXTERNAL_VERTEX_NUM */
    X->from = EXTERNAL_VERTEX_NUM;
```

```
    X->to   = EXTERNAL_VERTEX_NUM;

    arc_release(X->arc);            X->arc = NULL;
  }
}


/*************************************************************************
 * st_copy(STREAM X)                                              @16 *
 *                                                                    *
 * Deep copy of a Stream                                              *
 *************************************************************************/
#ifdef _NO_PROTO
STREAM st_copy(X)
STREAM X;
{
#else
STREAM st_copy(STREAM X) {
#endif

  STREAM temp_st;

  if (X != NULL) {
    temp_st = (STREAM) malloc(sizeof(ST_NODE));

    temp_st->id = X->id;
    temp_st->label = strdup(X->label);
    temp_st->label_font = X->label_font;
    temp_st->label_x_offset = X->label_x_offset;
    temp_st->label_y_offset = X->label_y_offset;
    temp_st->from = X->from;
    temp_st->to = X->to;
    temp_st->arc = arc_copy(X->arc);
    temp_st->latency = X->latency;
    temp_st->latency_unit = X->latency_unit;
    temp_st->latency_font = X->latency_font;
    temp_st->latency_x_offset = X->latency_x_offset;
    temp_st->latency_y_offset = X->latency_y_offset;
    temp_st->stream_type_name = strdup(X->stream_type_name);
    temp_st->state_initial_value = strdup(X->state_initial_value);
    temp_st->is_state_variable = X->is_state_variable;
    temp_st->is_new = X->is_new;
    temp_st->is_modified = X->is_modified;
    temp_st->is_deleted = X->is_deleted;

    return(temp_st);

  }
  else
    return(NULL);
}
```

```
/*************************************************************************
 * st_list_release(ST_LIST X)                                      @7 *
 *                                                                    *
 * Recovers all dynamic memory allocated to an ST_LIST.               *
 * Original ST_LIST pointer is set to NULL on return.                 *
 *************************************************************************/
#ifdef _NO_PROTO
void st_list_release(X)
ST_LIST X;
{
#else
void st_list_release(ST_LIST X) {
#endif

  ST_LIST tempST;

  while (X != NULL) {
    tempST = X->next;
    /* MTS 11/7/96 change st_release(&(X->st)) to st_release(X->st) */
    st_release(X->st);
    free((char *) X->st);
    free((char *) X);
    X = tempST;
  }
}


/*************************************************************************
 * st_list_copy(ST_LIST X)                                        @16   *
 *                                                                    *
 * Deep copy of an ST_LIST                                            *
 *************************************************************************/
#ifdef _NO_PROTO
ST_LIST st_list_copy(X)
ST_LIST X;
{
#else
ST_LIST st_list_copy(ST_LIST X) {
#endif

  ST_LIST tempST;

  if (X == NULL)
    return(NULL);
  else
  {
    tempST = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
    tempST->st = st_copy(X->st);
    tempST->next = st_list_copy(X->next);
    return(tempST);
  }
}
```

```
/*************************************************************************
 * graph_release(GRAPH_DESC x)                                        @7 *
 *                                                                       *
 * Recovers all dynamic memory allocated to a GRAPH_DESC_NODE.           *
 * Original pointer to GRAPH_DESC_NODE is untouched.  Only the memory    *
 * allocated inside the GRAPH_DESC_NODE is recovered.                    *
 *************************************************************************/
#ifdef _NO_PROTO
void graph_release(x)
GRAPH_DESC x;
{
#else
void graph_release(GRAPH_DESC x) {
#endif

  if (x != NULL) {
    free(x->root_op_name);              x->root_op_name    = NULL;
    free(x->parent_op_name);            x->parent_op_name  = NULL;
    free(x->current_op_name);           x->current_op_name = NULL;

    st_list_release(x->input_list);     x->input_list      = NULL;
    st_list_release(x->output_list);    x->output_list     = NULL;

    free(x->cur_op_spec);               x->cur_op_spec     = NULL;

    op_list_release(x->operator_list);  x->operator_list   = NULL;
    st_list_release(x->stream_list);    x->stream_list     = NULL;
    id_list_release(x->timer_list);     x->timer_list      = NULL;

    free(x->graph_informal_desc);       x->graph_informal_desc = NULL;

    id_list_release(x->avail_impl_langs); x->avail_impl_langs  = NULL;

    free(x->global_types);              x->global_types    = NULL;

  }
}


/*************************************************************************
 * err_msgs_release(ERROR_MSGS *ErrPtr)                                  *
 *                                                                       *
 * Recovers all dynamic memory allocated to an ERROR_MSGS list.          *
 * Original ERROR_MSGS pointer is set to NULL on return.                 *
 *************************************************************************/
#ifdef _NO_PROTO
void err_msgs_release(ErrPtr)
ERROR_MSGS *ErrPtr;
{
#else
void err_msgs_release(ERROR_MSGS *ErrPtr)
#endif
  ERROR_MSGS tempErr, X;

  if (ErrPtr != NULL) {
    X = *ErrPtr;

    while (X != NULL) {
      tempErr = X->next;
      free((char *) X->parent_op_label);
      free((char *) X->cur_op_label);
      free((char *) X->msg);
      free((char *) X);
      X = tempErr;
    }
    *ErrPtr = X;
  }
}

/*************************************************************************
 * time_unit_str(int time, int unit)                                     *
 *                                                                       *
 * This routine returns a pointer to a string which contains the time    *
 * and units based on the input parameters.                              *
 *************************************************************************/
#ifdef _NO_PROTO
char* time_with_units(time, unit)
int time;
int unit;
{
#else
char* time_with_units(int time, int unit) {
#endif

  char buffer[100];

  if (time == UNDEFINED_TIME)
    return dup_str("");

  switch(unit) {
    case US:
      sprintf(buffer, "%d us", time);
      break;
    case MS:
      sprintf(buffer, "%d ms", time);
      break;
    case SECOND:
      sprintf(buffer, "%d sec", time);
      break;
    case MINUTE:
      sprintf(buffer, "%d min", time);
      break;
    case HOUR:
      sprintf(buffer, "%d hr", time);
```

```
      break;
    default:
      sprintf(buffer, "%d illegal unit", time);
      break;
  }
  return dup_str(buffer);
}


/*************************************************************************
 * operator_id_ptr(OP_LIST ptr, int id)                          @3      *
 *                                                                       *
 * Given an OP_LIST and an ID, this routine will search the OP_LIST for  *
 * the first OPERATOR with a matching ID and return a pointer to that    *
 * OPERATOR.  If a matching ID cannot be found in OP_LIST, NULL is       *
 * returned.                                                             *
 *************************************************************************/
#ifdef _NO_PROTO
OPERATOR operator_id_ptr(ptr, id)
OP_LIST ptr;
int id;
{
#else
OPERATOR operator_id_ptr(OP_LIST ptr, int id) {
#endif

  while (ptr != NULL) {
    if (ptr->op->id == id)
      return ptr->op;
    ptr = ptr->next;
  }
  return NULL;
}

/*************************************************************************
 * dup_str(char* ptr)                                                    *
 *                                                                       *
 * This routine uses strdup to make a copy of the string located at ptr. *
 * However, this routine will return an empty string ("") if ptr is NULL.*
 *************************************************************************/

#ifdef _NO_PROTO
char* dup_str(ptr)
char* ptr;
{
#else
char* dup_str(char* ptr) {
#endif

  if (ptr == NULL)
    return strdup("");
  else
    return strdup(ptr);
}


#ifdef _NO_PROTO
void init_operator(OpPtr)
OPERATOR OpPtr;
{
#else
void init_operator(OPERATOR OpPtr) {
#endif

  if (OpPtr != NULL) {
    OpPtr->id                     = UNDEFINED_OPNUM;
    OpPtr->op_num                 = UNDEFINED_OPNUM;
    OpPtr->x                      = 0;
    OpPtr->y                      = 0;
    OpPtr->radius                 = 0;
    OpPtr->color                  = 0;
    OpPtr->label                  = dup_str("");
    OpPtr->label_font             = 0;
    OpPtr->label_x_offset         = 0;
    OpPtr->label_y_offset         = 0;
    OpPtr->timing_type            = NTC;
    OpPtr->met                    = UNDEFINED_TIME;
    OpPtr->met_unit               = MS;
    OpPtr->met_font               = 0;
    OpPtr->met_x_offset           = 0;
    OpPtr->met_y_offset           = 0;
    OpPtr->met_reqmts             = NULL;
    OpPtr->period                 = UNDEFINED_TIME;
    OpPtr->period_unit            = MS;
    OpPtr->period_reqmts          = NULL;
    OpPtr->fw                     = UNDEFINED_TIME;
    OpPtr->fw_unit                = MS;
    OpPtr->fw_reqmts              = NULL;
    OpPtr->mrt                    = UNDEFINED_TIME;
    OpPtr->mrt_unit               = MS;
    OpPtr->mrt_reqmts             = NULL;
    OpPtr->mcp                    = UNDEFINED_TIME;
    OpPtr->mcp_unit               = MS;
    OpPtr->mcp_reqmts             = NULL;
    OpPtr->trigger_type           = UNPROTECTED;
    OpPtr->trigger_set            = NULL;
    OpPtr->if_condition           = dup_str("");
    OpPtr->trigger_reqmts         = NULL;
    OpPtr->output_guard_list      = dup_str("");
    OpPtr->exception_list         = dup_str("");
    OpPtr->timer_op_list          = dup_str("");
    OpPtr->key_word_list          = NULL;
    OpPtr->operator_informal_desc = dup_str("");
    OpPtr->operator_formal_desc   = dup_str("");
    OpPtr->operator_impl_lang     = dup_str(DEFAULT_IMPL_LANG);
    OpPtr->is_composite           = false;
```

```
    OpPtr->is_terminator          = false;
    OpPtr->is_new                 = false;
    OpPtr->is_modified            = false;
    OpPtr->is_deleted             = false;
  }
}


#ifdef _NO_PROTO
void init_stream(StPtr)
STREAM StPtr;
{
#else
void init_stream(STREAM StPtr) {
#endif

  if (StPtr != NULL) {
    StPtr->id                     = UNDEFINED_OPNUM;
    StPtr->label                  = dup_str("");
    StPtr->label_font             = 0;
    StPtr->label_x_offset         = 0;
    StPtr->label_y_offset         = 0;
    /* MTS 11/7/96 change NULL to EXTERNAL_VERTEX_NUM */
    StPtr->from                   = EXTERNAL_VERTEX_NUM;
    StPtr->to                     = EXTERNAL_VERTEX_NUM;
    StPtr->arc                    = NULL;
    StPtr->latency                = UNDEFINED_TIME;
    StPtr->latency_unit           = MS;
    StPtr->latency_font           = 0;
    StPtr->latency_x_offset       = 0;
    StPtr->latency_y_offset       = 0;
    StPtr->stream_type_name       = dup_str("");
    StPtr->state_initial_value    = dup_str("");
    StPtr->is_state_variable      = false;
    StPtr->is_new                 = false;
    StPtr->is_modified            = false;
    StPtr->is_deleted             = false;
  }
}

#ifdef _NO_PROTO
void init_action(act)
ACTION act;
{
#else
void init_action(ACTION act);
#endif

  act->next_op = NULL;

}

#ifdef _NO_PROTO
```



```
void init_graph_desc(gdPtr)
GRAPH_DESC gdPtr;
{
#else
void init_graph_desc(GRAPH_DESC gdPtr) {
#endif

  if (gdPtr != NULL) {
    gdPtr->root_op_name           = dup_str("");
    gdPtr->root_op_num            = UNDEFINED_OPNUM;
    gdPtr->parent_op_name         = dup_str("");
    gdPtr->parent_op_num          = UNDEFINED_OPNUM;
    gdPtr->current_op_name        = dup_str("");
    gdPtr->current_op_num         = UNDEFINED_OPNUM;

    gdPtr->input_list             = NULL;
    gdPtr->output_list            = NULL;

    gdPtr->cur_op_spec            = NULL;

    gdPtr->cur_op_spec_met        = UNDEFINED_TIME;
    gdPtr->cur_op_spec_met_unit   = MS;
    gdPtr->cur_op_is_terminator   = false;

    gdPtr->operator_list          = NULL;
    gdPtr->stream_list            = NULL;

    gdPtr->timer_list             = NULL;
    gdPtr->graph_informal_desc    = dup_str("");

    gdPtr->avail_impl_langs       = NULL;

    gdPtr->global_types           = dup_str("");
  }
}

/*   Determines whether the input string represents a valid number. */

#ifdef _NO_PROTO
BOOLEAN valid_num_string(num)
char *num;
{
#else
BOOLEAN valid_num_string(char *num) {
#endif

  int index, num_length;

  if(num != NULL) {
    num_length =.strlen(num);
    if(num_length > 0) {
      for(index = 0; index < num_length; index++) {
        if((num[index] != '\n') &&
```

```
          (num[index] != '-') &&
          ((num[index] < '0') || (num[index] > '9')))
        return false;
      }
      return true;
    }
  }
  return false;
}


/*****************************************************************************
 * valid_integer_literal(char *num, int *value)                    @8        *
 *                                                                           *
 * Verify that a string (num) is a valid PSDL integer_literal, which has     *
 * no sign.  Leading and trailing whitespace is ignored.                     *
 *                                                                           *
 * integer_literal ::= digit {digit}                                         *
 * digit           ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9               *
 *****************************************************************************/
#ifdef _NO_PROTO
BOOLEAN valid_integer_literal(num, value)
char *num;
int *value;
{
#else
BOOLEAN valid_integer_literal(char *num, int *value) {
#endif

  int i, ix, num_length;

  BOOLEAN validated = false;

  if (num != NULL) {

    num_length = strlen(num);
    if (num_length > 0) {

      /* Skip over any leading whitespace */
      for (i = 0; ((i < num_length) && isspace(num[i])); i++)
;
      ix = i;

      if (ix < num_length) {
for (i = ix; ((i < num_length) && isdigit(num[i])); i++)
  ;
ix = i;

if (ix == num_length)
  validated = true;
else {
  /* Check that there is only trailing whitespace remaining */
  for (i = ix; ((i < num_length) && isspace(num[i])); i++)
    ;

  if (i == num_length)
    validated = true;
}
      }
    }
  }

  /* Get value of integer_literal */
  if (validated)
    *value = atoi(num);
  else
    *value = 0;

  return validated;
}


/*****************************************************************************
 * valid_id(char *id)                                              @11       *
 *                                                                           *
 * Verify that a string is a valid PSDL id.                                  *
 *                                                                           *
 * id            ::= letter {alpha_numeric}                                  *
 * alpha_numeric ::= letter | digit | _                                      *
 *****************************************************************************/
#ifdef _NO_PROTO
BOOLEAN valid_id(id)
char *id;
{
#else
BOOLEAN valid_id(char *id) {
#endif

  char *idPtr;

  idPtr = id;

  if (!isalpha(*idPtr))
    return false;

  idPtr++;
  while (*idPtr != '\0') {
    if (!(isalnum(*idPtr) || (*idPtr == '_')))
      return false;
    idPtr++;
  }

  return true;
}
```

```
/*****************************************************************************
 * parse_id()                                                          @14   *
 *                                                                           *
 * Pull out the valid part of an id, if no valid part, return false.         *
 *****************************************************************************/
#ifdef _NO_PROTO
BOOLEAN parse_id(ID, idLen, allow_types)
char *ID;
int  *idLen;
BOOLEAN allow_types;
{
#else
BOOLEAN parse_id(char *ID, int *idLen, BOOLEAN allow_types) {
#endif

  char    *tempID;
  char    *idPtr;
  BOOLEAN  cont_parsing, key;

  *idLen = 0;
  if (!ID)         /* If empty or NULL, it is not a valid op_id     */
    return false;
  if (*ID == '\0')
    return false;

  idPtr = ID;
  if (!isalpha(*idPtr))
    return false;

  idPtr++;
  cont_parsing = true;
  while ((*idPtr != '\0') && cont_parsing) {
    if (!(isalnum(*idPtr) || (*idPtr == '_')))
      cont_parsing = false;
    else
      idPtr++;
  }

  *idLen = (idPtr - ID);
  tempID = malloc((*idLen) + 1);
  strncpy(tempID, ID, *idLen);
  *(tempID+(*idLen)) = '\0';      /* terminate string */

  key = is_keyword(tempID, allow_types);
  free(tempID);

  return !key;
}


/*****************************************************************************
 * valid_op_id(char *opID)                                             @13   *
 *                                                                           *
 * Verify that a string is a valid PSDL op_id.  If the opID has any          *
 * whitespace within the parenthesis, this whitespace will be removed from   *
 * opID when returned.                                                       *
 *                                                                           *
 * op_id         ::= [id "."] op_name [ "(" [id_list] "|" [id_list] ")"]     *
 * op_name       ::= id                                                      *
 * id_list       ::= id {"," id}                                             *
 * id            ::= letter {alpha_numeric}                                  *
 * alpha_numeric ::= letter | digit | _                                      *
 *****************************************************************************/
#ifdef _NO_PROTO
BOOLEAN valid_op_id(opID)
char *opID;
{
#else
BOOLEAN valid_op_id(char *opID) {
#endif

#define OPID_INITIAL       0
#define OPID_ID            1
#define OPID_DOT           2
#define OPID_OPNAME        3
#define OPID_PARAMS        4
#define OPID_INPUT_LIST    5
#define OPID_INPUT_LIST_C  6
#define OPID_OUTPUTS       7
#define OPID_OUTPUT_LIST   8
#define OPID_OUTPUT_LIST_C 9
#define OPID_CLOSE         10
#define OPID_FINAL         11

  int state;  /* uses above definitions to track location within op_id */
  BOOLEAN validated, process_next;
  char *tempOpId, *OpIdPtr, *tempOpIdPtr;
  int idLen;  /* length of an id within the op_id */

  if (!opID || *opID == '\0')  /* If empty or NULL, it is not a valid op_id */
    return false;

  tempOpId = (char *) malloc(strlen(opID));  /* area to remove whitespace */

  state        = OPID_INITIAL;
  OpIdPtr      = opID;
  tempOpIdPtr  = tempOpId;
  validated    = true;    /* assume it is validated until proved otherwise */
  process_next = true;
  *tempOpIdPtr = '\0';

  while (process_next && validated) {
    switch (state) {
    case OPID_INITIAL:
      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
```

```
if (validated = parse_id(OpIdPtr, &idLen, false)) {
  strncat(tempOpIdPtr, OpIdPtr, idLen);
  tempOpIdPtr += idLen;
  OpIdPtr     += (idLen - 1); /* will increment at the end */
  state = OPID_ID;
}
else
  validated = false;
      }
      break;

      case OPID_ID:
if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
  if (!(*OpIdPtr))
    state = OPID_FINAL;
  else if (*OpIdPtr == '.') {
    state = OPID_DOT;
    *tempOpIdPtr = *OpIdPtr;
    tempOpIdPtr++;
    *tempOpIdPtr = '\0';   /* Need to end string */
  }
  else
    validated = false;
}

break;

    case OPID_DOT:
      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (validated = parse_id(OpIdPtr, &idLen, false)) {
  strncat(tempOpIdPtr, OpIdPtr, idLen);
  tempOpIdPtr += idLen;
  OpIdPtr     += (idLen - 1); /* will increment at the end */
  state = OPID_OPNAME;
}
else
  validated = false;
      }
      break;

    case OPID_OPNAME:
      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (!(*OpIdPtr))
  state = OPID_FINAL;
else if (*OpIdPtr == '(') {
  state = OPID_PARAMS;
  *tempOpIdPtr = *OpIdPtr;
  tempOpIdPtr++;
  *tempOpIdPtr = '\0';   /* Need to end string */
}
else
  validated = false;
      }
      break;

    case OPID_PARAMS:
      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (*OpIdPtr == '|') {
  state = OPID_OUTPUTS;
  *tempOpIdPtr = *OpIdPtr;
  tempOpIdPtr++;
  *tempOpIdPtr = '\0';   /* Need to end string */
}
else if (validated = parse_id(OpIdPtr, &idLen, false)) {
  strncat(tempOpIdPtr, OpIdPtr, idLen);
  tempOpIdPtr += idLen;
  OpIdPtr     += (idLen - 1); /* will increment at the end */
  state = OPID_INPUT_LIST;
}
else      /* already set invalid by parse_id, but just in case */
  validated = false;
      }

      break;

    case OPID_INPUT_LIST:

      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (*OpIdPtr == ',') {
  state = OPID_INPUT_LIST_C;
  *tempOpIdPtr = *OpIdPtr;
  tempOpIdPtr++;
  *tempOpIdPtr = '\0';   /* Need to end string */
}
else if (*OpIdPtr == '|') {
  state = OPID_OUTPUTS;
  *tempOpIdPtr = *OpIdPtr;
  tempOpIdPtr++;
  *tempOpIdPtr = '\0';   /* Need to end string */
}
else
  validated = false;
      }

      break;

    case OPID_INPUT_LIST_C:

      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (validated = parse_id(OpIdPtr, &idLen, false)) {
  strncat(tempOpIdPtr, OpIdPtr, idLen);
  tempOpIdPtr += idLen;
  OpIdPtr     += (idLen - 1); /* will increment at the end */
  state = OPID_INPUT_LIST;
}
else      /* already set invalid by parse_id, but just in case */
```

```
  validated = false;
      }

      break;

    case OPID_OUTPUTS:
      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (*OpIdPtr == ')') {
  state = OPID_CLOSE;
  *tempOpIdPtr = *OpIdPtr;
  tempOpIdPtr++;
  *tempOpIdPtr = '\0';   /* Need to end string */
}
else if (validated = parse_id(OpIdPtr, &idLen, false)) {
  strncat(tempOpIdPtr, OpIdPtr, idLen);
  tempOpIdPtr += idLen;
  OpIdPtr     += (idLen - 1); /* will increment at the end */
  state = OPID_OUTPUT_LIST;
}
else      /* already set invalid by parse_id, but just in case */
  validated = false;
      }

      break;

    case OPID_OUTPUT_LIST:

      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (*OpIdPtr == ',') {
  state = OPID_OUTPUT_LIST_C;
  *tempOpIdPtr = *OpIdPtr;
  tempOpIdPtr++;
  *tempOpIdPtr = '\0';   /* Need to end string */
}
else if (*OpIdPtr == ')') {
  state = OPID_CLOSE;
  *tempOpIdPtr = *OpIdPtr;
  tempOpIdPtr++;
  *tempOpIdPtr = '\0';   /* Need to end string */
}
else
  validated = false;
      }

      break;

    case OPID_OUTPUT_LIST_C:

      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (validated = parse_id(OpIdPtr, &idLen, false)) {
  strncat(tempOpIdPtr, OpIdPtr, idLen);
  tempOpIdPtr += idLen;
  OpIdPtr     += (idLen - 1); /* will increment at the end */
  state = OPID_OUTPUT_LIST;
}
else      /* already set invalid by parse_id, but just in case */
  validated = false;
      }

      break;

    case OPID_CLOSE:
      if (!((*OpIdPtr == ' ') || (*OpIdPtr == '\t'))) {
if (!(*OpIdPtr))
  state = OPID_FINAL;
else
  validated = false;
      }

      break;
    }
    if (!(*OpIdPtr))     /* no more input, do not loop any more  */
      process_next = false;
    else
      OpIdPtr++;
  }

  if (state != OPID_FINAL)   /* did we end correctly */
    validated = false;

  if (validated)
    strcpy(opID, tempOpId);   /* use op_id with whitespace removed */

  free(tempOpId);
  return validated;
}


/*****************************************************************************
 * valid_type_name(char *typeName)                                    @15    *
 *                                                                           *
 * Verify that a string is a valid PSDL type_name.  If the typeName has      *
 * any whitespace within the brackets, this whitespace will be removed       *
 * from the typeName when returned.                                          *
 *                                                                           *
 * type_name     ::= id | id "[" type_decl "]"                               *
 * type_decl     ::= id_list ":" type_name {"," id_list ":" type_name}       *
 * id_list       ::= id {"," id}                                             *
 * id            ::= letter {alpha_numeric}                                  *
 * alpha_numeric ::= letter | digit | _                                      *
 *****************************************************************************/
#ifdef _NO_PROTO
BOOLEAN valid_type_name(typeName)
char *typeName;
{
#else
```

```
BOOLEAN valid_type_name(char *typeName) {
#endif

  /* Definition of states */
#define TYPE_INITIAL       0
#define TYPE_VALID_ID      1
#define TYPE_DECL          2
#define TYPE_DECL_VALID_ID 3
#define TYPE_NAME          4
#define TYPE_NAME_VALID_ID 5
#define TYPE_NAME_CLOSE    6
#define TYPE_FINAL         7

  int state;  /* one of the above defined states */
  BOOLEAN validated, process_next;
  int idLen;
  char *locTypeName, *srcPtr, *locPtr, *typePtr;
  int level;   /* recursive type_decl */

  if (!typeName || *typeName == '\0')
    return false;

  locTypeName = malloc(strlen(typeName));   /* area to remove whitespace */

  state = TYPE_INITIAL;
  srcPtr = typeName;
  locPtr = locTypeName;
  validated    = true;        /* assume validated until proven otherwise */
  process_next = true;        /* input to process                        */
  *locPtr = '\0';
  level = 0;

  while (process_next && validated) {
    switch (state) {
    case TYPE_INITIAL:
      if (!((*srcPtr == ' ') || (*srcPtr == '\t'))) {
if (validated = parse_id(srcPtr, &idLen, true)) { /*check on types*/
  strncat(locPtr, srcPtr, idLen);
  typePtr = locPtr;      /* for future test of type */
  locPtr += idLen;
  srcPtr += (idLen -1);  /* will increment at the bottom */
  state = TYPE_VALID_ID;
}
      }
      break;

    case TYPE_VALID_ID:
      if (!((*srcPtr == ' ') || (*srcPtr == '\t'))) {
if (!(*srcPtr))
  state = TYPE_FINAL;
else if (*srcPtr == '[') {
  /* before we go any farther, make sure previous ID was not a type */
  if (validated = (!is_keyword(typePtr, false))) {
    *locPtr = *srcPtr;
    locPtr++;
    *locPtr = '\0';        /* terminate string for now */
    state = TYPE_DECL;
  }
  else
    validated = false;
}
else
  validated = false;
      }
      break;

    case TYPE_DECL:
      if (!((*srcPtr == ' ') || (*srcPtr == '\t'))) {
if (validated = parse_id(srcPtr, &idLen, false)) {
  strncat(locPtr, srcPtr, idLen);
  locPtr += idLen;
  srcPtr += (idLen -1);  /* will increment at the bottom */
  state = TYPE_DECL_VALID_ID;
}
      }
      break;

    case TYPE_DECL_VALID_ID:
      if (!((*srcPtr == ' ') || (*srcPtr == '\t'))) {
if (*srcPtr == ',' || *srcPtr == ':') {
  *locPtr = *srcPtr;
  locPtr++;
  *locPtr = '\0';        /* terminate string for now */
  state = (*srcPtr == ',') ? TYPE_DECL : TYPE_NAME;
}
else
  validated = false;
      }
      break;

    case TYPE_NAME:
      if (!((*srcPtr == ' ') || (*srcPtr == '\t'))) {
if (validated = parse_id(srcPtr, &idLen, true)) {
  strncat(locPtr, srcPtr, idLen);
  locPtr += idLen;
  srcPtr += (idLen -1);  /* will increment at the bottom */
  state = TYPE_NAME_VALID_ID;
}
      }
      break;

    case TYPE_NAME_VALID_ID:
      if (!((*srcPtr == ' ') || (*srcPtr == '\t'))) {
if (*srcPtr == ']' || *srcPtr == ',') {
  *locPtr = *srcPtr;
  locPtr++;
```

```
  *locPtr = '\0';        /* terminate string for now */
  state = (*srcPtr == ']') ? TYPE_NAME_CLOSE : TYPE_DECL;
} else if (*srcPtr == '[') {   /* recursive type_decl */
  level++;
  state = TYPE_DECL;
  *locPtr = *srcPtr;
  locPtr++;
  *locPtr = '\0';        /* terminate string for now */
}
else
  validated = false;
      }
      break;

    case TYPE_NAME_CLOSE:
      if (!((*srcPtr == ' ') || (*srcPtr == '\t'))) {
if (!(*srcPtr))
  state = TYPE_FINAL;
else if (level) {
  level--;
  state = TYPE_NAME_VALID_ID;
  srcPtr--;    /* decrement so that you can reprocess next char */
}
else
  validated = false;
      }
      break;
    }

    if (!(*srcPtr))           /* no more input, do not loop any more     */
      process_next = false;
    else
      srcPtr++;
  }

  if (state != TYPE_FINAL || level)
    validated = false;

  if (validated)
    strcpy(typeName, locTypeName);   /* return copy with whitespace removed */

  free(locTypeName);
  return validated;
}


/*****************************************************************************
 * is_keyword(char *id)                                              @12     *
 *                                                                           *
 * Return true if id is a keyword as defined by sde.abstract.ssl             *
 * Comparison is made case insensitive since the synthesiser generator       *
 * matches both upper and lower case (even though mixed characters are       *
 * not caught by the SG, they are still considered keywords.                 *
 *****************************************************************************/
#ifdef _NO_PROTO
BOOLEAN is_keyword(id, allow_types)
char *id;
BOOLEAN allow_types;
{
#else
BOOLEAN is_keyword(char *id, BOOLEAN allow_types) {
#endif

  /* Note that a slight improvement in performance can be made by stopping
     search when keywords are searched in alphabetical order.  Make sure
     that keywords are inserted in alphabetical order
     (use emacs <esc>-x sort-lines).                                        */
  const int max_keys = 43;      /* make sure you update number of keywords */

  typedef struct keyword_record {
    char    *id;
    BOOLEAN type;
  } keyword_record;

  static keyword_record keywords[] = {
    "ABS", false,
    "ALL", false,
    "AND", false,
    "AXIOMS", false,
    "BOOLEAN", true,
    "DESCRIPTION", false,
    "EDGE", false,
    "END", false,
    "EXCEPTION", false,
    "EXCEPTIONS", false,
    "EXTERNAL", false,
    "FALSE,", false,
    "GENERIC", false,
    "GRAPH", false,
    "HOURS", false,
    "IF", false,
    "IMPLEMENTATION", false,
    "INITIALLY", false,
    "INPUT", false,
    "INTEGER", true,
    "KEYWORDS", false,
    "MICROSEC", false,
    "MIN", false,
    "MOD", false,
    "MS", false,
    "NOT", false,
    "OPERATOR", false,
    "OR", false,
    "OUTPUT", false,
    "PERIOD", false,
    "PROPERTY", false,
```

```
    "REAL", true,
    "REM", false,
    "SEC", false,
    "SOME", false,
    "SPECIFICATION", false,
    "STATES", false,
    "TIMER", false,
    "TRIGGERED", false,
    "TRUE", false,
    "TYPE", false,
    "VERTEX", false,
    "XOR", false
  };

  int compare;
  int i;

  i = 0;
  while (( i < max_keys) &&
         ((compare = strcasecmp(id, keywords[i].id)) > 0) )
    i++;

  if (compare == 0)
    if (allow_types && keywords[i].type)
      return false;
    else
      return true;
  else
    return false;
}


/*************************************************************************
 * white_space(char *text)                                          @9   *
 *                                                                       *
 * Check to see if text is white space.  White space has been expanded to *
 * include a NULL pointer as well as space, tabs, etc.                   *
 *************************************************************************/
#ifdef _NO_PROTO
BOOLEAN white_space(text)
char *text;
{
#else
BOOLEAN white_space(char *text) {
#endif

  int i, text_length;

  BOOLEAN validated = true;

  if (text != NULL) {

    text_length = strlen(text);

    for (i = 0; ((i < text_length) && isspace(text[i])); i++)
;

    if (i < text_length)
      validated = false;
  }
  return validated;
}


/*************************************************************************
 * at_list_copy(AT_LIST y)                                          @10  *
 *                                                                       *
 * Based on id_list_copy written by Doug Lange 9/8/96.                   *
 * Builds and returns an AAT_LIST.                                       *
 * Deep copy of an AT_LIST.  The new AT_LIST may be modified in          *
 * that any AT_NODE with an attr of NULL will be returned with an        *
 * attr containing an empty string ("").                                 *
 *************************************************************************/
#ifdef _NO_PROTO
AT_LIST at_list_copy(y)
AT_LIST y;
{
#else
AT_LIST at_list_copy(AT_LIST y) {
#endif
  /* y is the original AT_LIST,
     x is the new AT_LIST,
     temp1 is used to build the new AT_LIST,
     temp2 is used to step through the original AT_LIST.        */
  AT_LIST temp1, temp2;

  if (y != NULL) {      /* build the new list */
    AT_LIST x = (AT_LIST) malloc(sizeof(AT_NODE));
    x->next = NULL;
    temp1 = x;
    temp2 = y;
    while (temp2 != NULL) {
      temp1->attr = dup_str(temp2->attr);
      temp1->reqmts = id_list_copy(temp2->reqmts);

      if (temp2->next != NULL) {
temp1->next = (AT_LIST) malloc(sizeof(AT_NODE));
temp1 = temp1->next;
      }
      temp1->next = NULL;
      temp2 = temp2->next;
    }
    return(x);
  }
  else
    return(NULL);
```

```
}


/************************************************************************
 * at_list_release(AT_LIST X)                                    @10  *
 *                                                                    *
 * based on the id_list_release written by Doug Lange 9/8/96.         *
 * Releases cells.                                                    *
 * Recovers all dynamic memory allocated to an AT_LIST.               *
 ************************************************************************/
#ifdef _NO_PROTO
void at_list_release(X)
AT_LIST X;
{
#else
void at_list_release(AT_LIST X) {
#endif
  AT_LIST  temp1;

  while (X != NULL) {     /* delete x's cells */
    temp1 = X->next;
    free(X->attr);
    id_list_release(X->reqmts);   X->reqmts = NULL;
    free((char *)X);
    X = temp1;              /* will return NULL on exit */
  }
  return;
}


/************************************************************************
 * at_list_replace(AT_LIST *x, AT_LIST y)                        @10  *
 *                                                                    *
 * based on id_list_replace written by Doug Lange 9/8/96.             *
 * Replaces what was in x with y while ensuring releasing cells.      *
 * This routine performs a deep copy replacement of one AT_LIST with  *
 * another AT_LIST.                                                   *
 ************************************************************************/
#ifdef _NO_PROTO
void at_list_replace(x, y)
AT_LIST *x;
AT_LIST y;
{
#else
void at_list_replace(AT_LIST *x, AT_LIST y) {
#endif

  if (x != NULL) {
    at_list_release(*x);
    *x = at_list_copy(y);
  }
  else {
#ifdef GE_DEBUG
  printf("ERROR:  at_list_replace to a NULL location.\n");
#endif
  }
}


/************************************************************************
 * at_list_search(char *label, AT_LIST y)                        @12  *
 *                                                                    *
 * returns pointer to the at_list_node containing the label           *
 * returns null if no such node exists                                *
 ************************************************************************/
#ifdef _NO_PROTO
AT_LIST at_list_search(label, y)
char *label;
AT_LIST y;
{
#else
AT_LIST at_list_search(char *label, AT_LIST y) {
#endif

  if (y == NULL)
    return(NULL);
  else if (strcmp(y->attr, label) == 0)
    return(y);
  else
    return(at_list_search(label, y->next));
}


#ifdef __cplusplus
}
#endif
```

```
/* ************************************************************

Name:          ge_utilities_debug.h

************************************************************ */
#ifndef GE_UTILITIES_DEBUG_H
#define GE_UTILITIES_DEBUG_H 1

#include <stdio.h>
#include "ge_interface.h"


#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

void write_gdn_file();
int  read_gdn_file();
void display_gdn();

/*******************************************************/

int  read_operator();
int  read_stream();
int  read_timer();
int  read_impl_langs();

ID_LIST read_ID_LIST();
SPLINE_PTR read_SPLINES();

void print_ID_LIST();
void print_SPLINE_LIST();
void print_OPERATOR_LIST();
void print_STREAM_LIST();

void fprint_ID_LIST();
void fprint_SPLINE_LIST();
void fprint_OPERATOR_LIST();
void fprint_STREAM_LIST();

int  set_stream_values();
int  set_operator_values();

void read_line();
int  read_quote();

void print_summary();
void summarize_ID_LIST();

OPERATOR operator_ptr();

int encode_TF();
int encode_timing();
int encode_trigger();
int encode_time();

#else

void write_gdn_file(GRAPH_DESC_NODE *gdn, FILE *fp);
int  read_gdn_file(GRAPH_DESC gdnPtr, FILE *fp);
void display_gdn(GRAPH_DESC_NODE *gdn);

/*******************************************************/

int  read_operator(GRAPH_DESC_NODE *gdn, FILE *fp);
int  read_stream(ST_LIST *StOut, OP_LIST OpPtr, FILE *fp);
int  read_timer(GRAPH_DESC_NODE *gdn, FILE *fp);
int  read_impl_langs(GRAPH_DESC_NODE *gdn, FILE *fp);

ID_LIST read_ID_LIST(FILE *fp);
SPLINE_PTR read_SPLINES(FILE *fp);

void print_ID_LIST(ID_LIST id);
void print_SPLINE_LIST(SPLINE_PTR spPtr);
void print_OPERATOR_LIST(OP_LIST opPtr);
void print_STREAM_LIST(ST_LIST stPtr);

void fprint_ID_LIST(ID_LIST id, FILE *fp);
void fprint_SPLINE_LIST(SPLINE_PTR spPtr, FILE *fp);
void fprint_OPERATOR_LIST(OP_LIST opPtr, FILE *fp);
void fprint_STREAM_LIST(ST_LIST stPtr, FILE *fp);

int  set_stream_values(STREAM StPtr, char *in, char *key,
       char *value, FILE *fp,
       OP_LIST OpPtr);
int  set_operator_values(OPERATOR OpPtr, char *in, char *key,
 char *value, FILE *fp);

void read_line(char *in, char **out);
int  read_quote(char *in, char **out, FILE *fp);

void print_summary(GRAPH_DESC gd);
void summarize_ID_LIST(ID_LIST idPtr);

OPERATOR operator_ptr(char *ID_str, GRAPH_DESC_NODE *gdn);

int encode_TF(char *value);
int encode_timing(char *value);
int encode_trigger(char *value);
int encode_time(char *units);

#endif

#ifdef __cplusplus
```

```
}
#endif
#endif
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "ge_interface.h"
#include "ge_interface_labels.h"
#include "ge_utilities.h"
#include "ge_utilities_debug.h"

#define ERROR_CODE    1
#define NO_ERROR 0
#define MAX_LINE 1000

#ifdef __cplusplus
extern "C" {
#endif

/*****************************************************************/

#ifdef _NO_PROTO
void write_gdn_file(gdn, fp)
GRAPH_DESC_NODE *gdn;
FILE *fp;
{
#else
void write_gdn_file(GRAPH_DESC_NODE *gdn, FILE *fp) {
#endif

  fprintf(fp,"#GRAPH_DESCRIPTION\n");
  fprintf(fp,"  root_op_name:\t\t\"%s\"\n",
  (gdn->root_op_name) ? gdn->root_op_name : "");
  fprintf(fp,"  root_op_num:\t\t%d\n",gdn->root_op_num);
  fprintf(fp,"  parent_op_name:\t\"%s\"\n",
  (gdn->parent_op_name) ? gdn->parent_op_name : "");
  fprintf(fp,"  parent_op_num:\t%d\n",gdn->parent_op_num);
  fprintf(fp,"  current_op_name:\t\"%s\"\n",
  (gdn->current_op_name) ? gdn->current_op_name : "");
  fprintf(fp,"  current_op_num:\t%d\n",gdn->current_op_num);
  fprintf(fp,"  graph_informal_desc:\t\"%s\"\n",
 (gdn->graph_informal_desc) ? gdn->graph_informal_desc : "");
  fprintf(fp,"  global_types:\t\t\"%s\"\n",
 (gdn->global_types) ? gdn->global_types : "");

  fprintf(fp,"  op_spec_met:\t\t%d\n",gdn->cur_op_spec_met);
  fprintf(fp,"  op_spec_met_units:\t%s\n",
  time_units[(gdn->cur_op_spec_met_unit)+1]);
  fprintf(fp,"  cur_op_is_terminator:\t%s\n",
  (gdn->cur_op_is_terminator) ? "True" : "False");

  /* cur_op_spec_generic */
  /* cur_op_spec_input */
  /* cur_op_spec_output */
  /* cur_op_spec_states */
  /* cur_op_spec_exceptions */

  fprintf(fp,"\n  #OPERATOR_LIST\t%s\n",
  (gdn->operator_list) ? "" : "NULL");
  fprint_OPERATOR_LIST(gdn->operator_list, fp);
  fprintf(fp,"  #OPERATOR_LIST_END\n");

  fprintf(fp,"\n  #STREAM_LIST\t%s\n",
  (gdn->stream_list) ? "" : "NULL");
  fprint_STREAM_LIST(gdn->stream_list, fp);
  fprintf(fp,"  #STREAM_LIST_END\n");

  fprintf(fp,"\n  #TIMER_LIST\t%s\n",
  (gdn->timer_list) ? "" : "NULL");
  if (gdn->timer_list) fprint_ID_LIST(gdn->timer_list, fp);
  fprintf(fp,"  #TIMER_LIST_END\n");

  fprintf(fp,"\n  #INPUT_LIST\t%s\n",
  (gdn->input_list) ? "" : "NULL");
  fprint_STREAM_LIST(gdn->input_list, fp);
  fprintf(fp,"  #STREAM_LIST_END\n");

  fprintf(fp,"\n  #OUTPUT_LIST\t%s\n",
  (gdn->output_list) ? "" : "NULL");
  fprint_STREAM_LIST(gdn->output_list, fp);
  fprintf(fp,"  #STREAM_LIST_END\n");

  fprintf(fp,"\n  #IMPL_LANG_LIST\t%s\n",
  (gdn->avail_impl_langs) ? "" : "NULL");
  if (gdn->avail_impl_langs) fprint_ID_LIST(gdn->avail_impl_langs, fp);
  fprintf(fp,"  #IMPL_LANG_LIST_END\n");

  fprintf(fp,"\n#GRAPH_END\n");
}

/*****************************************************************/

#ifdef _NO_PROTO
int read_gdn_file(gdnPtr, fp)
GRAPH_DESC gdnPtr;
FILE *fp;
{
#else
int read_gdn_file(GRAPH_DESC gdnPtr, FILE *fp) {
#endif

  char in[MAX_LINE];
  char key[MAX_LINE];
  char param[MAX_LINE];
  int  error;

  graph_release(gdnPtr);
  init_graph_desc(gdnPtr);
```

```
  /* Read file until start of data */
  do {
    *key = ' ';
    if (!fgets(in,MAX_LINE,fp) || (sscanf(in,"%s",key)==EOF)) {
      printf("ERROR: No data found.\n");
      return ERROR_CODE;
    }
  } while (strcmp(key,"#GRAPH_DESCRIPTION")!=0);

  error = NO_ERROR;
  while (fgets(in,MAX_LINE,fp) && (error == NO_ERROR)) {
    *key = ' ';
    if (sscanf(in,"%s", key) != EOF) {
      if      (strcmp(key,"current_op_name:") == 0)
error = read_quote(in,&gdnPtr->current_op_name,fp);
      else if (strcmp(key,"current_op_num:") == 0) {
sscanf(in,"%s%s", key, param);
gdnPtr->current_op_num = atoi(param);
      }
      else if (strcmp(key,"parent_op_name:") == 0)
error = read_quote(in,&gdnPtr->parent_op_name,fp);
      else if (strcmp(key,"parent_op_num:") == 0) {
sscanf(in,"%s%s", key, param);
gdnPtr->parent_op_num = atoi(param);
      }
      else if (strcmp(key,"root_op_name:") == 0)
error = read_quote(in,&gdnPtr->root_op_name,fp);
      else if (strcmp(key,"root_op_num:") == 0) {
sscanf(in,"%s%s", key, param);
gdnPtr->root_op_num = atoi(param);
      }
      else if (strcmp(key,"op_spec_met:") == 0) {
sscanf(in,"%s%s", key, param);
gdnPtr->cur_op_spec_met = atoi(param);
      }
      else if (strcmp(key,"op_spec_met_units:") == 0) {
sscanf(in,"%s%s", key, param);
gdnPtr->cur_op_spec_met_unit = encode_time(param);
      }
      else if (strcmp(key,"cur_op_is_terminator:")==0)
gdnPtr->cur_op_is_terminator = encode_TF(param);
      else if (strcmp(key,"graph_informal_desc:") == 0)
error = read_quote(in,&gdnPtr->graph_informal_desc,fp);
      else if (strcmp(key,"global_types:") == 0)
error = read_quote(in,&gdnPtr->global_types,fp);
      else if (strcmp(key,"#OPERATOR_LIST") == 0)
error = read_operator(gdnPtr,fp);
      else if (strcmp(key,"#STREAM_LIST") == 0)
error = read_stream(&gdnPtr->stream_list,gdnPtr->operator_list,fp);
      else if (strcmp(key,"#TIMER_LIST") == 0)
error = read_timer(gdnPtr,fp);
      else if (strcmp(key,"#INPUT_LIST") == 0)
error = read_stream(&gdnPtr->input_list,NULL,fp);
      else if (strcmp(key,"#OUTPUT_LIST") == 0)
error = read_stream(&gdnPtr->output_list,NULL,fp);
      else if (strcmp(key,"#IMPL_LANG_LIST") == 0)
error = read_impl_langs(gdnPtr,fp);
      else if (strcmp(key,"#GRAPH_END") == 0)
return NO_ERROR;
    }
  }
  printf("WARNING:  Problem was encountered while reading data.\n");
  return ERROR_CODE;  /* Something went wrong, pass on what we have */
}

/*************************************************************/

#ifdef _NO_PROTO
void display_gdn(gdn)
GRAPH_DESC_NODE *gdn;
{
#else
void display_gdn(GRAPH_DESC_NODE *gdn) {
#endif

  printf("GRAPH DESCRIPTION:\n");
  printf("  root_op_name:      \"%s\"\n",
 (gdn->root_op_name) ? gdn->root_op_name : "");
  printf("  parent_op_name:      \"%s\"\n",
 (gdn->parent_op_name) ? gdn->parent_op_name : "");
  printf("  current_op_name:     \"%s\"\n",
 (gdn->current_op_name) ? gdn->current_op_name : "");

  printf("  graph_informal_desc: \"%s\"\n",
 (gdn->graph_informal_desc) ? gdn->graph_informal_desc : "");

  printf("  global_types:\t\t\"%s\"\n",
 (gdn->global_types) ? gdn->global_types : "");

  printf("  op_spec_met:\t\t%d\n",gdn->cur_op_spec_met);
  printf("  op_spec_met_units:\t%s\n",
  time_units[(gdn->cur_op_spec_met_unit)+1]);
  printf("  cur_op_is_terminator:\t%s\n",
  (gdn->cur_op_is_terminator) ? "True" : "False");

  printf("\n  OPERATOR LIST: %s\n",
 (gdn->operator_list) ? "" : "NULL");
  print_OPERATOR_LIST(gdn->operator_list);

  printf("\n  STREAM LIST:   %s\n",
 (gdn->stream_list) ? "" : "NULL");
  print_STREAM_LIST(gdn->stream_list);

  printf("\n  TIMER LIST:    %s\n",
 (gdn->timer_list) ? "" : "NULL");
```

```
  if (gdn->timer_list) print_ID_LIST(gdn->timer_list);

  printf("\n  INPUT LIST:   %s\n",
 (gdn->input_list) ? "" : "NULL");
  print_STREAM_LIST(gdn->input_list);

  printf("\n  OUTPUT LIST:   %s\n",
 (gdn->output_list) ? "" : "NULL");
  print_STREAM_LIST(gdn->output_list);

  printf("\n  IMPL_LANG_LIST\t%s\n",
  (gdn->avail_impl_langs) ? "" : "NULL");
  if (gdn->avail_impl_langs) print_ID_LIST(gdn->avail_impl_langs);

  printf("GRAPH END.\n");
}

/*****************************************************************/


#ifdef _NO_PROTO
void summarize_ID_LIST(idPtr)
ID_LIST idPtr;
{
#else
void summarize_ID_LIST(ID_LIST idPtr) {
#endif

  printf("    ID_LIST: ");
  while (idPtr != NULL) {
    printf("%s, ", idPtr->id);
    idPtr = idPtr->next;
  }
  printf("\n");
}

/*****************************************************************/

#ifdef _NO_PROTO
void summarize_SPLINE_LIST(spPtr)
SPLINE_PTR spPtr;
{
#else
void summarize_SPLINE_LIST(SPLINE_PTR spPtr) {
#endif

  while (spPtr != NULL) {
    printf(" %d %d,", spPtr->x, spPtr->y);
    spPtr = spPtr->next;
  }
}

/*****************************************************************/
```



```
#ifdef _NO_PROTO
void summarize_STREAM_LIST(stPtr)
ST_LIST stPtr;
{
#else
void summarize_STREAM_LIST(ST_LIST stPtr) {
#endif
  while (stPtr != NULL) {
    printf("    Stream ID %d %s : ", stPtr->st->id, stPtr->st->label);

    summarize_SPLINE_LIST(stPtr->st->arc);
    printf("\n");

    stPtr = stPtr->next;
  }
}

/*****************************************************************/
#ifdef _NO_PROTO
void summarize_OPERATOR_LIST(opPtr)
OP_LIST opPtr;
{
#else
void summarize_OPERATOR_LIST(OP_LIST opPtr) {
#endif
  while (opPtr != NULL) {
    printf("    Operator ID %d %s \n", opPtr->op->id, opPtr->op->label);

    opPtr = opPtr->next;
  }
}

/*****************************************************************/

#ifdef _NO_PROTO
void print_summary(gd)
GRAPH_DESC gd;
{
#else
void print_summary(GRAPH_DESC gd) {
#endif

  printf("GRAPH DESCRIPTION:\n");
  printf("  root_op_name:      \"%s\"\n",
 (gd->root_op_name) ? gd->root_op_name : "");
  printf("  parent_op_name:      \"%s\"\n",
 (gd->parent_op_name) ? gd->parent_op_name : "");
  printf("  current_op_name:     \"%s\"\n",
 (gd->current_op_name) ? gd->current_op_name : "");

  printf("  graph_informal_desc: \"%s\"\n",
 (gd->graph_informal_desc) ? gd->graph_informal_desc : "");
```

```
 /*
 printf("  op_spec:\t\t\"%s\"\n",
(gd->cur_op_spec) ? gd->cur_op_spec : "");

 printf("  global_types:\t\t\"%s\"\n",
(gd->global_types) ? gd->global_types : "");
 */

 printf("\n  INPUT LIST:   %s\n",
(gd->input_list) ? "" : "NULL");
 summarize_STREAM_LIST(gd->input_list);

 printf("\n  OUTPUT LIST:   %s\n",
(gd->output_list) ? "" : "NULL");
 summarize_STREAM_LIST(gd->output_list);

 printf("\n  OPERATOR LIST: %s\n",
(gd->operator_list) ? "" : "NULL");
 summarize_OPERATOR_LIST(gd->operator_list);

 printf("\n  STREAM LIST:   %s\n",
(gd->stream_list) ? "" : "NULL");
 summarize_STREAM_LIST(gd->stream_list);

 printf("\n  TIMER LIST:    %s\n",
(gd->timer_list) ? "" : "NULL");
 if (gd->timer_list) summarize_ID_LIST(gd->timer_list);

 printf("\n  IMPL_LANG_LIST\t%s\n",
 (gd->avail_impl_langs) ? "" : "NULL");
 if (gd->avail_impl_langs) summarize_ID_LIST(gd->avail_impl_langs);

 printf("GRAPH END.\n");
}

/*****************************************************************/

#ifdef _NO_PROTO
void read_line(in, out)
char *in;
char **out;
{
#else
void read_line(char *in, char **out) {
#endif

  char  key[MAX_LINE];
  char  value[MAX_LINE];
  char  start[MAX_LINE],rest[MAX_LINE];
  char  *new_line;
  char  *out_str;

  sscanf(in,"%s%s%1000c",key,start,rest);
  if (new_line = strstr(rest,"\n"))  /* Strip out newline */
    *new_line = '\0';

  out_str = dup_str(strcat(start,rest));
  *out = out_str;
}

/*****************************************************************/

#ifdef _NO_PROTO
int read_quote(in, out, fp)
char *in;
char **out;
FILE *fp;
{
#else
int read_quote(char *in, char **out, FILE *fp) {
#endif

  char  *start_quote, *end_quote;
  char  value[MAX_LINE];
  char  result[MAX_LINE];
  char  *res_str = result;
  char  *out_str;

  if (start_quote = strchr(in,'\"')) {
    start_quote++;
    if (end_quote = strstr(start_quote,"\"")) {
      *end_quote = '\0';
      out_str = dup_str(start_quote);
      *out = out_str;
      return NO_ERROR;
    }
    else {
      *res_str = '\0';
      strcpy(res_str,start_quote);

      if (fgets(in,MAX_LINE,fp) != 0) {
while ((end_quote = strstr(in,"\"")) == NULL) {
  strcat(res_str,in);
  if (fgets(in,MAX_LINE,fp) == 0)
    return ERROR_CODE;
}
*end_quote = '\0';
strcat(res_str,in);
out_str = dup_str(res_str);
*out = out_str;
return NO_ERROR;
      }
      else
return ERROR_CODE;
    }
```

```
  }
  else
    return ERROR_CODE;
}

/*****************************************************************/

#ifdef _NO_PROTO
int read_operator(gdn, fp)
GRAPH_DESC_NODE *gdn;
FILE *fp;
{
#else
int read_operator(GRAPH_DESC_NODE *gdn, FILE *fp) {
#endif

  OP_LIST   OpList, tail;
  OPERATOR  OpPtr;

  char in[MAX_LINE];
  char key[MAX_LINE];
  char value[MAX_LINE];

  int  error;
  int  repeat;
  int  scnCnt;

  gdn->operator_list = NULL;

  error = NO_ERROR;
  while ((error == NO_ERROR) && (fgets(in,MAX_LINE,fp) != 0)) {
    *key = ' '; *value = ' ';
    scnCnt = sscanf(in,"%s%1000c",key,value);
    if (strcmp(key,"#OPERATOR_LIST_END") == 0) {
      return NO_ERROR;
    }
    else if (key[0] == '#') {   /* New list started, we have a problem */
      return ERROR_CODE;
    }
    else { /* Read operators */
      if (strcmp(key,"begin_operator")) {
OpList = (OP_LIST) malloc(sizeof(OP_LIST_NODE));
OpPtr  = (OPERATOR)  malloc(sizeof(OP_NODE));
init_operator(OpPtr);

if (gdn->operator_list == NULL)
  gdn->operator_list = OpList;
else
  tail->next = OpList;

OpList->op   = OpPtr;
OpList->next = NULL;
tail = OpList;

repeat = 1;
do {
  if (strcmp(key,"end_operator") !=0) {
    error = set_operator_values(OpPtr, in, key, value, fp);
    if (fgets(in,MAX_LINE,fp) != 0) {
      *key = ' '; *value = ' ';
      scnCnt = sscanf(in,"%s%1000c",key,value);
    }
    else {
      error = ERROR_CODE;
    }
  }
  else             /* Done with this operator, any more? */
    repeat = 0;
} while (repeat && (error == NO_ERROR));
      }
    }
  }
  return ERROR_CODE;  /* Something went wrong, did not find END key */
}

/*****************************************************************/

#ifdef _NO_PROTO
int read_stream(StOutP, OpPtr, fp)
ST_LIST *StOutP;
OP_LIST OpPtr;
FILE *fp;
{
#else
int read_stream(ST_LIST *StOutP, OP_LIST OpPtr, FILE *fp) {
#endif

  ST_LIST StOut;
  ST_LIST StList, tail;
  STREAM  StPtr;

  char in[MAX_LINE];
  char key[MAX_LINE];
  char value[MAX_LINE];

  int  error;
  int  repeat;
  int  scnCnt;

  /* Assumes that StOutP has already been released */
  StOut = NULL;
  *StOutP = StOut;

  error = NO_ERROR;
  while ((error == NO_ERROR) && (fgets(in,MAX_LINE,fp) != 0)) {
    *key = ' '; *value = ' ';
```

```
    scnCnt = sscanf(in,"%s%1000c",key,value);
    if (strcmp(key,"#STREAM_LIST_END") == 0) {
      return NO_ERROR;
    }
    else if (key[0] == '#') {   /* New list started, we have a problem */
      return ERROR_CODE;
    }
    else { /* Read streams */
      if (scnCnt > 0) {
StList = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
StPtr  = (STREAM)  malloc(sizeof(ST_NODE));
init_stream(StPtr);

if (StOut == NULL) {
  StOut   = StList;
  *StOutP = StOut;
}
else
  tail->next = StList;

StList->st   = StPtr;
StList->next = NULL;
tail = StList;

repeat = 1;
do {
  if (strcmp(key,"end_stream") !=0) {
    error = set_stream_values(StPtr, in, key, value, fp, OpPtr);
    if (fgets(in,MAX_LINE,fp) != 0) {
      *key = ' ';  *value = ' ';
      sscanf(in,"%s%1000c",key,value);
    }
    else {
      error = ERROR_CODE;
    }
  }
  else
    repeat = 0;
} while (repeat && (error == NO_ERROR));
      }
    }
  }
  return ERROR_CODE;
}

/*****************************************************************/

#ifdef _NO_PROTO
int read_timer(gdn, fp)
GRAPH_DESC_NODE *gdn;
FILE *fp;
{
#else
int read_timer(GRAPH_DESC_NODE *gdn, FILE *fp) {
#endif

  ID_LIST timer, tail;

  char in[MAX_LINE];
  char value[MAX_LINE];
  int  repeat;
  int  error;
  int  scnCnt;

  gdn->timer_list = NULL;

  error = NO_ERROR;
  while ((error == NO_ERROR) && (fgets(in,MAX_LINE,fp) != 0)) {
    *value = ' ';
    scnCnt = sscanf(in,"%s",value);
    if ((strcmp(value,"#TIMER_LIST_END") == 0) ||
(strcmp(value,"end_id_list")     == 0))   {
      return NO_ERROR;
    }
    else if (value[0] == '#') {   /* New list started, we have a problem */
      return ERROR_CODE;
    }
    else { /* Add timer value onto list of timers */
      if (scnCnt > 0) {
timer = (ID_LIST) malloc(sizeof(ID_NODE));
if (gdn->timer_list == NULL)
  gdn->timer_list = timer;
else
  tail->next = timer;
timer->id = dup_str(value);
timer->next = NULL;
tail = timer;
      }
    }
  }
  return ERROR_CODE;
}
/*****************************************************************/

#ifdef _NO_PROTO
int read_impl_langs(gdn, fp)
GRAPH_DESC_NODE *gdn;
FILE *fp;
{
#else
int read_impl_langs(GRAPH_DESC_NODE *gdn, FILE *fp) {
#endif

  ID_LIST lang, tail;

  char in[MAX_LINE];
```

```
  char value[MAX_LINE];
  int  repeat;
  int  error;
  int  scnCnt;

  gdn->avail_impl_langs = NULL;

  error = NO_ERROR;
  while ((error == NO_ERROR) && (fgets(in,MAX_LINE,fp) != 0)) {
    *value = ' ';
    scnCnt = sscanf(in,"%s",value);
    if ((strcmp(value,"#IMPL_LANG_LIST_END") == 0) ||
(strcmp(value,"end_id_list")     == 0))   {
      return NO_ERROR;
    }
    else if (value[0] == '#') {   /* New list started, we have a problem */
      return ERROR_CODE;
    }
    else {
      if (scnCnt > 0) {
lang = (ID_LIST) malloc(sizeof(ID_NODE));
if (gdn->avail_impl_langs == NULL)
  gdn->avail_impl_langs = lang;
else
  tail->next = lang;
lang->id = dup_str(value);
lang->next = NULL;
tail = lang;
      }
    }
  }
  return ERROR_CODE;
}

/*****************************************************************/

#ifdef _NO_PROTO
int encode_timing(value)
char *value;
{
#else
int encode_timing(char *value) {
#endif

  int i;

  for (i = 0; i < 3; i++) {
    if (strcmp(value,timing_types[i])==0)
      return (i);
  }
  return 0;
}
```



```
/*****************************************************************/

#ifdef _NO_PROTO
int encode_time(units)
char *units;
{
#else
int encode_time(char *units) {
#endif

  int i;

  for (i = 0; i < 6; i++) {
    if (strcmp(units,time_units[i])==0)
      return (i-1);
  }
  return -1;
}

/*****************************************************************/

#ifdef _NO_PROTO
int encode_trigger(value)
char *value;
{
#else
int encode_trigger(char *value) {
#endif

  int i;

  for (i = 0; i < 3; i++) {
    if (strcmp(value,trigger_conds[i])==0)
      return (i);
  }
  return 0;
}

/*****************************************************************/

#ifdef _NO_PROTO
int encode_TF(value)
char *value;
{
#else
int encode_TF(char *value) {
#endif
  if (strcmp(value,"True") == 0)
    return 1;
  return 0;
}

/*****************************************************************/
```

```
/*****************************************************************/

#ifdef _NO_PROTO
int set_stream_values(StPtr, in, key, value, fp, OpPtr)
STREAM StPtr;
char *in;
char *key;
char *value;
FILE *fp;
OP_LIST OpPtr;
{
#else
int set_stream_values(STREAM StPtr, char *in,  char *key,
      char *value, FILE *fp,
      OP_LIST OpPtr) {
#endif
  /* NOTE: Operators must be read in first in order to set up
           pointers for streams (from, to).                    */

  char temp_name[MAX_LINE];
  char param[MAX_LINE];
  char *name = temp_name;

  int error = NO_ERROR;

  *param = ' ';
  sscanf(value,"%s",param);

  if (strcmp(key,lb_id)==0)
    StPtr->id = atoi(param);
  else if (strcmp(key,lb_label)==0)
    error = read_quote(value,&StPtr->label,fp);
  else if (strcmp(key,lb_label_font)==0)
    StPtr->label_font = atoi(param);
  else if (strcmp(key,lb_label_x_offset)==0)
    StPtr->label_x_offset = atoi(param);
  else if (strcmp(key,lb_label_y_offset)==0)
    StPtr->label_y_offset = atoi(param);
  else if (strcmp(key,lb_from)==0)
    StPtr->from = atoi(param);
  else if (strcmp(key,lb_to)==0)
    StPtr->to = atoi(param);
  else if (strcmp(key,lb_latency)==0)
    StPtr->latency = atoi(param);
  else if (strcmp(key,lb_latency_unit)==0)
    StPtr->latency_unit = encode_time(param);
  else if (strcmp(key,lb_latency_font)==0)
    StPtr->latency_font = atoi(param);
  else if (strcmp(key,lb_latency_x_offset)==0)
    StPtr->latency_x_offset = atoi(param);
  else if (strcmp(key,lb_latency_y_offset)==0)
    StPtr->latency_y_offset = atoi(param);
  else if (strcmp(key,lb_stream_type_name)==0)
    error = read_quote(value,&StPtr->stream_type_name,fp);
  else if (strcmp(key,lb_state_initial_value)==0)
    error = read_quote(value,&StPtr->state_initial_value,fp);
  else if (strcmp(key,lb_is_state_variable)==0)
    StPtr->is_state_variable = encode_TF(param);
  else if (strcmp(key,lb_is_new)==0)
    StPtr->is_new = encode_TF(param);
  else if (strcmp(key,lb_is_modified)==0)
    StPtr->is_modified = encode_TF(param);
  else if (strcmp(key,lb_is_deleted)==0)
    StPtr->is_deleted = encode_TF(param);
  else if (strcmp(key,lb_arc)==0) {
    if (strstr(value,"NULL") != NULL)
      StPtr->arc = NULL;
    else
      StPtr->arc = read_SPLINES(fp);
  }
  return error;
}

/*****************************************************************/

#ifdef _NO_PROTO
int set_operator_values(OpPtr, in, key, value, fp)
OPERATOR OpPtr;
char *in;
char *key;
char *value;
FILE *fp;
{
#else
int set_operator_values(OPERATOR OpPtr, char *in,  char *key,
char *value, FILE *fp) {
#endif

  char temp_name[MAX_LINE];
  char param[MAX_LINE];
  char *name = temp_name;

  int error = NO_ERROR;

  *param = ' ';
  sscanf(value,"%s",param);

  if (strcmp(key,lb_id)==0)
    OpPtr->id = atoi(param);
  else if (strcmp(key,lb_op_num)==0)
    OpPtr->op_num = atoi(param);
  else if (strcmp(key,lb_x)==0)
    OpPtr->x = atoi(param);
  else if (strcmp(key,lb_y)==0)
```

```
  OpPtr->y = atoi(param);
else if (strcmp(key,lb_radius)==0)
  OpPtr->radius = atoi(param);
else if (strcmp(key,lb_color)==0)
  OpPtr->color = atoi(param);
else if (strcmp(key,lb_label)==0)
  error = read_quote(value,&OpPtr->label,fp);
else if (strcmp(key,lb_label_font)==0)
  OpPtr->label_font = atoi(param);
else if (strcmp(key,lb_label_x_offset)==0)
  OpPtr->label_x_offset = atoi(param);
else if (strcmp(key,lb_label_y_offset)==0)
  OpPtr->label_y_offset = atoi(param);
else if (strcmp(key,lb_timing_type)==0)
  OpPtr->timing_type = encode_timing(param);
else if (strcmp(key,lb_met)==0)
  OpPtr->met = atoi(param);
else if (strcmp(key,lb_met_unit)==0)
  OpPtr->met_unit = encode_time(param);
else if (strcmp(key,lb_met_font)==0)
  OpPtr->met_font = atoi(param);
else if (strcmp(key,lb_met_x_offset)==0)
  OpPtr->met_x_offset = atoi(param);
else if (strcmp(key,lb_met_y_offset)==0)
  OpPtr->met_y_offset = atoi(param);
else if (strcmp(key,lb_met_reqmts)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->met_reqmts = NULL;
  else
    OpPtr->met_reqmts = read_ID_LIST(fp);
else if (strcmp(key,lb_period)==0)
  OpPtr->period = atoi(param);
else if (strcmp(key,lb_period_unit)==0)
  OpPtr->period_unit = encode_time(param);
else if (strcmp(key,lb_period_reqmts)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->period_reqmts = NULL;
  else
    OpPtr->period_reqmts = read_ID_LIST(fp);
else if (strcmp(key,lb_fw)==0)
  OpPtr->fw = atoi(param);
else if (strcmp(key,lb_fw_unit)==0)
  OpPtr->fw_unit = encode_time(param);
else if (strcmp(key,lb_fw_reqmts)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->fw_reqmts = NULL;
  else
    OpPtr->fw_reqmts = read_ID_LIST(fp);
else if (strcmp(key,lb_mrt)==0)
  OpPtr->mrt = atoi(param);
else if (strcmp(key,lb_mrt_unit)==0)
  OpPtr->mrt_unit = encode_time(param);
else if (strcmp(key,lb_mrt_reqmts)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->mrt_reqmts = NULL;
  else
    OpPtr->mrt_reqmts = read_ID_LIST(fp);
else if (strcmp(key,lb_mcp)==0)
  OpPtr->mcp = atoi(param);
else if (strcmp(key,lb_mcp_unit)==0)
  OpPtr->mcp_unit = encode_time(param);
else if (strcmp(key,lb_mcp_reqmts)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->mcp_reqmts = NULL;
  else
    OpPtr->mcp_reqmts = read_ID_LIST(fp);
else if (strcmp(key,lb_trigger_type)==0)
  OpPtr->trigger_type = encode_trigger(param);
else if (strcmp(key,lb_trigger_set)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->trigger_set = NULL;
  else
    OpPtr->trigger_set = read_ID_LIST(fp);
else if (strcmp(key,lb_if_condition)==0)
  error = read_quote(in,&OpPtr->if_condition,fp);
else if (strcmp(key,lb_trigger_reqmts)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->trigger_reqmts = NULL;
  else
    OpPtr->trigger_reqmts = read_ID_LIST(fp);
else if (strcmp(key,lb_output_guard_list)==0)
  error = read_quote(in,&OpPtr->output_guard_list,fp);
else if (strcmp(key,lb_exception_list)==0)
  error = read_quote(in,&OpPtr->exception_list,fp);
else if (strcmp(key,lb_timer_op_list)==0)
  error = read_quote(in,&OpPtr->timer_op_list,fp);
else if (strcmp(key,lb_key_word_list)==0)
  if (strstr(value,"NULL") != NULL)
    OpPtr->key_word_list = NULL;
  else
    OpPtr->key_word_list = read_ID_LIST(fp);
else if (strcmp(key,lb_operator_informal_desc)==0)
  error = read_quote(in,&OpPtr->operator_informal_desc,fp);
else if (strcmp(key,lb_operator_formal_desc)==0)
  error = read_quote(in,&OpPtr->operator_formal_desc,fp);
else if (strcmp(key,lb_is_composite)==0)
  OpPtr->is_composite = encode_TF(param);
else if (strcmp(key,lb_is_terminator)==0)
  OpPtr->is_terminator = encode_TF(param);
else if (strcmp(key,lb_is_new)==0)
  OpPtr->is_new = encode_TF(param);
else if (strcmp(key,lb_is_modified)==0)
  OpPtr->is_modified = encode_TF(param);
else if (strcmp(key,lb_is_deleted)==0)
  OpPtr->is_deleted = encode_TF(param);
else if (strcmp(key,lb_operator_impl_lang)==0)
```

```
    error = read_quote(value,&OpPtr->operator_impl_lang,fp);

  return error;
}

/*****************************************************************/

#ifdef _NO_PROTO
ID_LIST read_ID_LIST(fp)
FILE *fp;
{
#else
ID_LIST read_ID_LIST(FILE *fp) {
#endif

  ID_LIST head, tail, node;

  char in[MAX_LINE];
  char value[MAX_LINE];
  int  repeat;
  int  error;
  int  scnCnt;

  head = NULL;

  error = NO_ERROR;
  while ((error == NO_ERROR) && (fgets(in,MAX_LINE,fp) != 0)) {
    *value = ' ';
    scnCnt = sscanf(in,"%s",value);
    if (strcmp(value,"end_id_list") == 0) {
      return head;
    }
    else { /* Add value onto list */
      if (scnCnt > 0) {
node = (ID_LIST) malloc(sizeof(ID_NODE));
if (head == NULL)
  head = node;
else
  tail->next = node;
node->id = dup_str(value);
node->next = NULL;
tail = node;
      }
    }
  }
  /* An error has occured */
  printf("Error in reading ID_LIST.\n");
  return NULL;
}

/*****************************************************************/

#ifdef _NO_PROTO
```

```
void print_ID_LIST(idPtr)
ID_LIST idPtr;
{
#else
void print_ID_LIST(ID_LIST idPtr) {
#endif

  while (idPtr != NULL) {
    printf("\t\t%s\n", idPtr->id);
    idPtr = idPtr->next;
  }
}

/*****************************************************************/

#ifdef _NO_PROTO
void fprint_ID_LIST(idPtr, fp)
ID_LIST idPtr;
FILE *fp;
{
#else
void fprint_ID_LIST(ID_LIST idPtr, FILE *fp) {
#endif

  while (idPtr != NULL) {
    fprintf(fp,"\t\t%s\n", idPtr->id);
    idPtr = idPtr->next;
  }
  fprintf(fp,"\tend_id_list\n");
}

/*****************************************************************/

#ifdef _NO_PROTO
SPLINE_PTR read_SPLINES(fp)
FILE *fp;
{
#else
SPLINE_PTR read_SPLINES(FILE *fp) {
#endif

  SPLINE_PTR head, tail, node;

  char in[MAX_LINE];
  char value[MAX_LINE];
  char label[MAX_LINE];
  char x_str[MAX_LINE],y_str[MAX_LINE];

  int  repeat;
  int  error;
  int  scnCnt;

  head = NULL;
```

```
  error = NO_ERROR;
  while ((error == NO_ERROR) && (fgets(in,MAX_LINE,fp) != 0)) {
    *value = ' ';
    scnCnt = sscanf(in,"%s%s%s%s",value,x_str,label,y_str);
    if (strcmp(value,"end_splines") == 0) {
      return head;
    }
    else { /* Add value onto list */
      if (scnCnt > 0) {
node = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
if (head == NULL)
  head = node;
else
  tail->next = node;
node->x    = atoi(x_str);
node->y    = atoi(y_str);
node->next = NULL;
tail = node;
      }
    }
  }
  /* An error has occured */
  printf("Error in reading SPLINE nodes.\n");
  return NULL;
}

/*****************************************************************/

#ifdef _NO_PROTO
void print_SPLINE_LIST(spPtr)
SPLINE_PTR spPtr;
{
#else
void print_SPLINE_LIST(SPLINE_PTR spPtr) {
#endif

  while (spPtr != NULL) {
    printf("\t\tx %d\ty %d\n", spPtr->x, spPtr->y);
    spPtr = spPtr->next;
  }
}

/*****************************************************************/

#ifdef _NO_PROTO
void fprint_SPLINE_LIST(spPtr, fp)
SPLINE_PTR spPtr;
FILE *fp;
{
#else
void fprint_SPLINE_LIST(SPLINE_PTR spPtr, FILE *fp) {
#endif

  while (spPtr != NULL) {
    fprintf(fp,"\t\tx %d\ty %d\n", spPtr->x, spPtr->y);
    spPtr = spPtr->next;
  }
  fprintf(fp,"\tend_splines\n");
}

/*****************************************************************/

#ifdef _NO_PROTO
void print_OPERATOR_LIST(opPtr)
OP_LIST opPtr;
{
#else
void print_OPERATOR_LIST(OP_LIST opPtr) {
#endif
  while (opPtr != NULL) {
    printf("\t%s\t\t\t%d\n",lb_id,
   opPtr->op->id);
    printf("\t%s\t\t\t%d\n",lb_op_num,
   opPtr->op->op_num);
    printf("\t%s\t\t\t%d\n",lb_x,
   opPtr->op->x);
    printf("\t%s\t\t\t%d\n",lb_y,
   opPtr->op->y);
    printf("\t%s\t\t\t%d\n",lb_radius,
   opPtr->op->radius);
    printf("\t%s\t\t\t%d\n",lb_color,
   opPtr->op->color);
    printf("\t%s\t\t\t\"%s\"\n",lb_label,
   (opPtr->op->label) ? opPtr->op->label : "");
    printf("\t%s\t\t%d\n",lb_label_font,
   opPtr->op->label_font);
    printf("\t%s\t\t%d\n",lb_label_x_offset,
   opPtr->op->label_x_offset);
    printf("\t%s\t\t%d\n",lb_label_y_offset,
   opPtr->op->label_y_offset);
    printf("\t%s\t\t%s\n",lb_timing_type,
   timing_types[opPtr->op->timing_type]);
    printf("\t%s\t\t\t%d\n",lb_met,
   opPtr->op->met);
    printf("\t%s\t\t%s\n",lb_met_unit,
   time_units[opPtr->op->met_unit+1]);
    printf("\t%s\t\t%d\n",lb_met_font,
   opPtr->op->met_font);
    printf("\t%s\t\t%d\n",lb_met_x_offset,
   opPtr->op->met_x_offset);
    printf("\t%s\t\t%d\n",lb_met_y_offset,
   opPtr->op->met_y_offset);
    printf("\t%s\t\t%s\n",lb_met_reqmts,
   (opPtr->op->met_reqmts) ? "" : "NULL");
    if (opPtr->op->met_reqmts) print_ID_LIST(opPtr->op->met_reqmts);
```

```
 printf("\t%s\t\t\t%d\n",lb_period,
opPtr->op->period);
 printf("\t%s\t\t%s\n",lb_period_unit,
time_units[opPtr->op->period_unit+1]);
 printf("\t%s\t\t%s\n",lb_period_reqmts,
(opPtr->op->period_reqmts) ? "" : "NULL");
 if (opPtr->op->period_reqmts) print_ID_LIST(opPtr->op->period_reqmts);
 printf("\t%s\t\t\t%d\n",lb_fw,
opPtr->op->fw);
 printf("\t%s\t\t\t%s\n",lb_fw_unit,
time_units[opPtr->op->fw_unit+1]);
 printf("\t%s\t\t%s\n",lb_fw_reqmts,
(opPtr->op->fw_reqmts) ? "" : "NULL");
 if (opPtr->op->fw_reqmts) print_ID_LIST(opPtr->op->fw_reqmts);
 printf("\t%s\t\t\t%d\n",lb_mrt,
opPtr->op->mrt);
 printf("\t%s\t\t%s\n",lb_mrt_unit,
time_units[opPtr->op->mrt_unit+1]);
 printf("\t%s\t\t%s\n",lb_mrt_reqmts,
(opPtr->op->mrt_reqmts) ? "" : "NULL");
 if (opPtr->op->mrt_reqmts) print_ID_LIST(opPtr->op->mrt_reqmts);
 printf("\t%s\t\t\t%d\n",lb_mcp,
opPtr->op->mcp);
 printf("\t%s\t\t%s\n",lb_mcp_unit,
time_units[opPtr->op->mcp_unit+1]);
 printf("\t%s\t\t%s\n",lb_mcp_reqmts,
(opPtr->op->mcp_reqmts) ? "" : "NULL");
 if (opPtr->op->mcp_reqmts) print_ID_LIST(opPtr->op->mcp_reqmts);
 printf("\t%s\t\t%s\n",lb_trigger_type,
trigger_conds[opPtr->op->trigger_type]);
 printf("\t%s\t\t%s\n",lb_trigger_set,
(opPtr->op->trigger_set) ? "" : "NULL");
 if (opPtr->op->trigger_set) print_ID_LIST(opPtr->op->trigger_set);
 printf("\t%s\t\t\"%s\"\n",lb_if_condition,
(opPtr->op->if_condition) ? opPtr->op->if_condition : "");
 printf("\t%s\t\t%s\n",lb_trigger_reqmts,
(opPtr->op->trigger_reqmts) ? "" : "NULL");
 if (opPtr->op->trigger_reqmts) print_ID_LIST(opPtr->op->trigger_reqmts);
 printf("\t%s\t\"%s\"\n",lb_output_guard_list,
(opPtr->op->output_guard_list) ?
  opPtr->op->output_guard_list : "");
 printf("\t%s\t\t\"%s\"\n",lb_exception_list,
(opPtr->op->exception_list) ?
  opPtr->op->exception_list : "");
 printf("\t%s\t\t\"%s\"\n",lb_timer_op_list,
(opPtr->op->timer_op_list) ?
  opPtr->op->timer_op_list : "");
 printf("\t%s\t\t%s\n",lb_key_word_list,
(opPtr->op->key_word_list) ? "" : "NULL");
 if (opPtr->op->key_word_list) print_ID_LIST(opPtr->op->key_word_list);
 printf("\t%s\t\"%s\"\n",lb_operator_informal_desc,
(opPtr->op->operator_informal_desc) ?
  opPtr->op->operator_informal_desc : "");
 printf("\t%s\t\"%s\"\n",lb_operator_formal_desc,
(opPtr->op->operator_formal_desc) ?
  opPtr->op->operator_formal_desc : "");
 printf("\t%s\t\t%s\n",lb_is_composite,
(opPtr->op->is_composite)      ? "True" : "False");
 printf("\t%s\t\t%s\n",lb_is_terminator,
(opPtr->op->is_terminator)     ? "True" : "False");
 printf("\t%s\t\t\t%s\n",lb_is_new,
(opPtr->op->is_new)            ? "True" : "False");
 printf("\t%s\t\t%s\n",lb_is_modified,
(opPtr->op->is_modified)       ? "True" : "False");
 printf("\t%s\t\t%s\n",lb_is_deleted,
(opPtr->op->is_deleted)        ? "True" : "False");
 printf("\t%s\t\t\t\"%s\"\n",lb_operator_impl_lang,
(opPtr->op->operator_impl_lang) ? opPtr->op->operator_impl_lang : "");
 printf("\n");

  opPtr = opPtr->next;
 }
}

/*****************************************************************/

#ifdef _NO_PROTO
void fprint_OPERATOR_LIST(opPtr, fp)
OP_LIST opPtr;
FILE *fp;
{
#else
void fprint_OPERATOR_LIST(OP_LIST opPtr, FILE *fp) {
#endif

 int first = 0;

 while (opPtr != NULL) {
  if (first) fprintf(fp,"\n");
  first = 1;
  fprintf(fp,"\tbegin_operator\n");
  fprintf(fp,"\t%s\t\t\t%d\n",lb_id,
  opPtr->op->id);
  fprintf(fp,"\t%s\t\t\t%d\n",lb_op_num,
  opPtr->op->op_num);
  fprintf(fp,"\t%s\t\t\t%d\n",lb_x,
  opPtr->op->x);
  fprintf(fp,"\t%s\t\t\t%d\n",lb_y,
  opPtr->op->y);
  fprintf(fp,"\t%s\t\t\t%d\n",lb_radius,
  opPtr->op->radius);
  fprintf(fp,"\t%s\t\t\t%d\n",lb_color,
  opPtr->op->color);
  fprintf(fp,"\t%s\t\t\t\"%s\"\n",lb_label,
  (opPtr->op->label) ? opPtr->op->label : "");
  fprintf(fp,"\t%s\t\t%d\n",lb_label_font,
```

```
opPtr->op->label_font);
fprintf(fp,"\t%s\t\t%d\n",lb_label_x_offset,
opPtr->op->label_x_offset);
fprintf(fp,"\t%s\t\t%d\n",lb_label_y_offset,
opPtr->op->label_y_offset);
fprintf(fp,"\t%s\t\t%s\n",lb_timing_type,
timing_types[opPtr->op->timing_type]);
fprintf(fp,"\t%s\t\t\t%d\n",lb_met,
opPtr->op->met);
fprintf(fp,"\t%s\t\t%s\n",lb_met_unit,
time_units[opPtr->op->met_unit+1]);
fprintf(fp,"\t%s\t\t%d\n",lb_met_font,
opPtr->op->met_font);
fprintf(fp,"\t%s\t\t%d\n",lb_met_x_offset,
opPtr->op->met_x_offset);
fprintf(fp,"\t%s\t\t%d\n",lb_met_y_offset,
opPtr->op->met_y_offset);
fprintf(fp,"\t%s\t\t%s\n",lb_met_reqmts,
(opPtr->op->met_reqmts) ? "" : "NULL");
if (opPtr->op->met_reqmts) fprint_ID_LIST(opPtr->op->met_reqmts, fp);
fprintf(fp,"\t%s\t\t\t%d\n",lb_period,
opPtr->op->period);
fprintf(fp,"\t%s\t\t%s\n",lb_period_unit,
time_units[opPtr->op->period_unit+1]);
fprintf(fp,"\t%s\t\t%s\n",lb_period_reqmts,
(opPtr->op->period_reqmts) ? "" : "NULL");
if (opPtr->op->period_reqmts) fprint_ID_LIST(opPtr->op->period_reqmts, fp);
fprintf(fp,"\t%s\t\t\t%d\n",lb_fw,
opPtr->op->fw);
fprintf(fp,"\t%s\t\t\t%s\n",lb_fw_unit,
time_units[opPtr->op->fw_unit+1]);
fprintf(fp,"\t%s\t\t%s\n",lb_fw_reqmts,
(opPtr->op->fw_reqmts) ? "" : "NULL");
if (opPtr->op->fw_reqmts) fprint_ID_LIST(opPtr->op->fw_reqmts, fp);
fprintf(fp,"\t%s\t\t\t%d\n",lb_mrt,
opPtr->op->mrt);
fprintf(fp,"\t%s\t\t%s\n",lb_mrt_unit,
time_units[opPtr->op->mrt_unit+1]);
fprintf(fp,"\t%s\t\t%s\n",lb_mrt_reqmts,
(opPtr->op->mrt_reqmts) ? "" : "NULL");
if (opPtr->op->mrt_reqmts) fprint_ID_LIST(opPtr->op->mrt_reqmts, fp);
fprintf(fp,"\t%s\t\t\t%d\n",lb_mcp,
opPtr->op->mcp);
fprintf(fp,"\t%s\t\t%s\n",lb_mcp_unit,
time_units[opPtr->op->mcp_unit+1]);
fprintf(fp,"\t%s\t\t%s\n",lb_mcp_reqmts,
(opPtr->op->mcp_reqmts) ? "" : "NULL");
if (opPtr->op->mcp_reqmts) fprint_ID_LIST(opPtr->op->mcp_reqmts, fp);
fprintf(fp,"\t%s\t\t%s\n",lb_trigger_type,
trigger_conds[opPtr->op->trigger_type]);
fprintf(fp,"\t%s\t\t%s\n",lb_trigger_set,
(opPtr->op->trigger_set) ? "" : "NULL");
if (opPtr->op->trigger_set) fprint_ID_LIST(opPtr->op->trigger_set, fp);
 fprintf(fp,"\t%s\t\t\"%s\"\n",lb_if_condition,
(opPtr->op->if_condition) ? opPtr->op->if_condition : "");
 fprintf(fp,"\t%s\t\t%s\n",lb_trigger_reqmts,
 (opPtr->op->trigger_reqmts) ? "" : "NULL");
 if (opPtr->op->trigger_reqmts) fprint_ID_LIST(opPtr->op->trigger_reqmts, fp);
 fprintf(fp,"\t%s\t\"%s\"\n",lb_output_guard_list,
(opPtr->op->output_guard_list) ?
   opPtr->op->output_guard_list : "");
 fprintf(fp,"\t%s\t\t\"%s\"\n",lb_exception_list,
(opPtr->op->exception_list) ?
   opPtr->op->exception_list : "");
 fprintf(fp,"\t%s\t\t\"%s\"\n",lb_timer_op_list,
(opPtr->op->timer_op_list) ?
   opPtr->op->timer_op_list : "");
 fprintf(fp,"\t%s\t\t%s\n",lb_key_word_list,
 (opPtr->op->key_word_list) ? "" : "NULL");
 if (opPtr->op->key_word_list) fprint_ID_LIST(opPtr->op->key_word_list, fp);
 fprintf(fp,"\t%s\t\"%s\"\n",lb_operator_informal_desc,
(opPtr->op->operator_informal_desc) ?
   opPtr->op->operator_informal_desc : "");
 fprintf(fp,"\t%s\t\"%s\"\n",lb_operator_formal_desc,
(opPtr->op->operator_formal_desc) ?
   opPtr->op->operator_formal_desc : "");
 fprintf(fp,"\t%s\t\t%s\n",lb_is_composite,
 (opPtr->op->is_composite)      ? "True" : "False");
 fprintf(fp,"\t%s\t\t%s\n",lb_is_terminator,
 (opPtr->op->is_terminator)     ? "True" : "False");
 fprintf(fp,"\t%s\t\t%s\n",lb_is_new,
 (opPtr->op->is_new)            ? "True" : "False");
 fprintf(fp,"\t%s\t\t%s\n",lb_is_modified,
 (opPtr->op->is_modified)       ? "True" : "False");
 fprintf(fp,"\t%s\t\t%s\n",lb_is_deleted,
 (opPtr->op->is_deleted)        ? "True" : "False");
 fprintf(fp,"\t%s\t\t\t\"%s\"\n",lb_operator_impl_lang,
(opPtr->op->operator_impl_lang) ? opPtr->op->operator_impl_lang : "");
 fprintf(fp,"\tend_operator\n");

 opPtr = opPtr->next;
  }
}

/*****************************************************************/

#ifdef _NO_PROTO
void print_STREAM_LIST(stPtr)
ST_LIST stPtr;
{
#else
void print_STREAM_LIST(ST_LIST stPtr) {
#endif
  while (stPtr != NULL) {
    printf("\t%s\t\t\t%d\n",lb_id,
   stPtr->st->id);
```

```
    printf("\t%s\t\t\t\"%s\"\n",lb_label,
   (stPtr->st->label) ? stPtr->st->label : "");
    printf("\t%s\t\t%d\n",lb_label_font,
   stPtr->st->label_font);
    printf("\t%s\t\t%d\n",lb_label_x_offset,
   stPtr->st->label_x_offset);
    printf("\t%s\t\t%d\n",lb_label_y_offset,
   stPtr->st->label_y_offset);
    printf("\t%s\t\t\t%d\n",lb_from,
   stPtr->st->from);
    printf("\t%s\t\t\t%d\n",lb_to,
   stPtr->st->to);
    printf("\t%s\t\t\t%d\n",lb_latency,
   stPtr->st->latency);
    printf("\t%s\t\t%s\n",lb_latency_unit,
   time_units[(stPtr->st->latency_unit)+1]);
    printf("\t%s\t\t%d\n",lb_latency_font,
   stPtr->st->latency_font);
    printf("\t%s\t%d\n",lb_latency_x_offset,
   stPtr->st->latency_x_offset);
    printf("\t%s\t%d\n",lb_latency_y_offset,
   stPtr->st->latency_y_offset);
    printf("\t%s\t\"%s\"\n",lb_stream_type_name,
   (stPtr->st->stream_type_name) ?
     stPtr->st->stream_type_name : "");
    printf("\t%s\t\"%s\"\n",lb_state_initial_value,
   (stPtr->st->state_initial_value) ?
     stPtr->st->state_initial_value : "");
    printf("\t%s\t%s\n",lb_is_state_variable,
   (stPtr->st->is_state_variable) ? "True" : "False");
    printf("\t%s\t\t\t%s\n",lb_is_new,
   (stPtr->st->is_new)            ? "True" : "False");
    printf("\t%s\t\t%s\n",lb_is_modified,
   (stPtr->st->is_modified)       ? "True" : "False");
    printf("\t%s\t\t%s\n",lb_is_deleted,
   (stPtr->st->is_deleted)        ? "True" : "False");
    printf("\t%s\t\t\t%s\n",lb_arc,
   (stPtr->st->arc) ? "" : "NULL");
    print_SPLINE_LIST(stPtr->st->arc);
    printf("\n");

    stPtr = stPtr->next;
  }
}

/*****************************************************************/

#ifdef _NO_PROTO
void fprint_STREAM_LIST(stPtr, fp)
ST_LIST stPtr;
FILE *fp;
{
#else
void fprint_STREAM_LIST(ST_LIST stPtr, FILE *fp) {
#endif

  int first = 0;

  while (stPtr != NULL) {
    if (first) fprintf(fp,"\n");
    first = 1;
    fprintf(fp,"\tbegin_stream\n");
    fprintf(fp,"\t%s\t\t\t%d\n",lb_id,
    stPtr->st->id);
    fprintf(fp,"\t%s\t\t\t\"%s\"\n",lb_label,
    (stPtr->st->label) ? stPtr->st->label : "");
    fprintf(fp,"\t%s\t\t%d\n",lb_label_font,
    stPtr->st->label_font);
    fprintf(fp,"\t%s\t\t%d\n",lb_label_x_offset,
    stPtr->st->label_x_offset);
    fprintf(fp,"\t%s\t\t%d\n",lb_label_y_offset,
    stPtr->st->label_y_offset);
    fprintf(fp,"\t%s\t\t\t%d\n",lb_from,
    stPtr->st->from);
    fprintf(fp,"\t%s\t\t\t%d\n",lb_to,
    stPtr->st->to);
    fprintf(fp,"\t%s\t\t\t%d\n",lb_latency,
    stPtr->st->latency);
    fprintf(fp,"\t%s\t\t%s\n",lb_latency_unit,
    time_units[(stPtr->st->latency_unit)+1]);
    fprintf(fp,"\t%s\t\t%d\n",lb_latency_font,
    stPtr->st->latency_font);
    fprintf(fp,"\t%s\t%d\n",lb_latency_x_offset,
    stPtr->st->latency_x_offset);
    fprintf(fp,"\t%s\t%d\n",lb_latency_y_offset,
    stPtr->st->latency_y_offset);
    fprintf(fp,"\t%s\t\"%s\"\n",lb_stream_type_name,
    (stPtr->st->stream_type_name) ?
      stPtr->st->stream_type_name : "");
    fprintf(fp,"\t%s\t\"%s\"\n",lb_state_initial_value,
    (stPtr->st->state_initial_value) ?
      stPtr->st->state_initial_value : "");
    fprintf(fp,"\t%s\t%s\n",lb_is_state_variable,
    (stPtr->st->is_state_variable) ? "True" : "False");
    fprintf(fp,"\t%s\t\t\t%s\n",lb_is_new,
            (stPtr->st->is_new)            ? "True" : "False");
    fprintf(fp,"\t%s\t\t%s\n",lb_is_modified,
    (stPtr->st->is_modified)       ? "True" : "False");
    fprintf(fp,"\t%s\t\t%s\n",lb_is_deleted,
    (stPtr->st->is_deleted)        ? "True" : "False");
    fprintf(fp,"\t%s\t\t\t%s\n",lb_arc,
    (stPtr->st->arc) ? "" : "NULL");
    fprint_SPLINE_LIST(stPtr->st->arc, fp);
    fprintf(fp,"\tend_stream\n");

    stPtr = stPtr->next;
```

```
  }
}

#ifdef __cplusplus
```

```
}
#endif
```

```
#ifndef get_unique_id_h
#define get_unique_id_h 1

#include <stdlib.h>
#include <stdio.h>


#ifdef __cplusplus
extern "C" {
#endif

extern void init_id_server();
extern unsigned int get_unique_id();

#ifdef __cplusplus
}
#endif

#endif
```

```
#include <stdlib.h>
#include <stdio.h>

#include "get_unique_id.h"

#ifdef __cplusplus
extern "C" {
#endif

void init_id_server() {
  FILE *id_file;

  unsigned int unique_id;

  if ((id_file = fopen("unique_id.dat", "r")) == NULL) {
    id_file = fopen("unique_id.dat", "w");
    if (id_file != NULL) {
      fprintf(id_file, "%d\n", 1);
      fclose(id_file);
    }
    else {
      printf("Unable to open the file unique_id.dat\n");
      clearerr(id_file);
    }
    unique_id = 1;
  }
  else {
    fscanf(id_file, "%d", &unique_id);
    fclose(id_file);
  }
}


unsigned int get_unique_id() {
  FILE *id_file;
  unsigned int temp_id, unique_id;

  id_file = fopen("unique_id.dat", "r");
  if (id_file != NULL) {
    fscanf(id_file, "%d", &unique_id);
    fclose(id_file);

    id_file = fopen("unique_id.dat", "w");
    if (id_file != NULL) {
      temp_id = unique_id + 1;
      fprintf(id_file, "%d\n", temp_id);
      fclose(id_file);
    }
    else {
      printf("Unable to open the file unique_id.dat for write\n");
      clearerr(id_file);
      unique_id = 1;
    }
  }
  else {
    printf("Unable to open the file unique_id.dat\n");
    clearerr(id_file);
    unique_id = 1;
  }
  return(unique_id);
}


#ifdef __cplusplus
}
#endif
```

```
/* Written by Dan Heller.  Copyright 1991, O'Reilly && Associates.
 * This program is freely distributable without licensing fees and
 * is provided without guarantee or warrantee expressed or implied.
 * This program is -not- in the public domain.
 */

#ifndef GETTOPSHELL_H
#define GETTOPSHELL_H

#include <Xm/DialogS.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

/* climb widget tree until we get to the top.  Return the Shell */
extern Widget GetTopShell() ;

#else

extern Widget GetTopShell(Widget w) ;

#endif /* _NO_PROTO */

#ifdef __cplusplus
}  /* Close scope of 'extern "C"' declaration which encloses file. */
#endif

#endif /* GETTOPSHELL_H */
```

```
/* Written by Dan Heller.  Copyright 1991, O'Reilly && Associates.
 * This program is freely distributable without licensing fees and
 * is provided without guarantee or warrantee expressed or implied.
 * This program is -not- in the public domain.
 */

#include <stdio.h>
#include <ctype.h>
#include <Xm/DialogS.h>

#include "gettopshell.h"

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

/* climb widget tree until we get to the top.  Return the Shell */
Widget
GetTopShell(w)
Widget w;

#else

Widget
GetTopShell(Widget w)

#endif /* _NO_PROTO */
{
    while (w && !XtIsWMShell(w))
        w = XtParent(w);
    return w;
};

#ifdef __cplusplus
}  /* Close scope of 'extern "C"' declaration which encloses file. */
#endif
```

```
/* ************************************************************

Name:     graph_editor.h
Author:   Lange and Anunciado
Program:  graph_editor
Remarks:

History:
  @1   96/10/01  Ken Moeller
       Upgraded calling arguments to reflect changes to requirements.

  @2   96/10/11  Ken Moeller
       Moved print command over to an XEvent using a global print
       command buffer.

************************************************************ */
#include "ge_interface.h"
#include "windows.h"
#include <Xm/Text.h>

#ifndef GRAPH_EDITOR_H
#define GRAPH_EDITOR_H 1

#define RING_BELL 1
```

```
static BOOLEAN   return_sde_flag;   // flag to break out of motif loop
static BOOLEAN   motif_initialized = false; // only do motif init once
static PrintBuf  PrintCmd;                                  // @2
static char      *prev_status;

void help_cb(Widget w, XtPointer client_data, XtPointer call_data);
void save_state(int);
void update_status(char* status, BOOLEAN bell);
void clear_status();

extern "C" {
int edit_graph(
       /* in out parameter */
       GRAPH_DESC current_graph,

       /* out parameter */
       ACTION next_action,

       /* in parameter */
       ERROR_MSGS sde_error_msgs);

}
#endif
```

```
/* ************************************************************

Name:          graph_editor.C
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 21 Sep 92
Remarks:    graph_editor.C is the main program for the
          CAPS '93 graph editor.  Depending on the command line
          parameters, it allows either viewing only or full
          editing of a graph passed by the CAPS '93 syntax-
          directed editor.

General Comments:

            The XmProcessTraversal function is called numerous
          places in an attempt to keep the keyboard input focus
          in the drawing window.  This allows the editor to
          respond to the delete and backspace key.  This works
          with varying degrees of success.

Credits:    Portions of code are adapted from the following:
          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.

Reengineering:
        Modified by Doug Lange on 8/12/96
                Removed Property button and put in its place a Timers button
        Modified by Doug Lange on 8/16/96 - 8/20/96
                Added callback and dialog for Timer Tool button.
        Modified by Doug Lange on 8/19/96
                Added callback and dialog for Informal Description Tool button.

History:
  @1   96/09/29  Ken Moeller
       Migration from Motif 1.2 to Motif 1.1.

  @2   96/10/01  Ken Moeller
       Upgraded calling arguments to reflect changes to requirements.

  @3   96/10/03  Ken Moeller
       Started to switch over to build_from_sde and write_to_sde.
       This is not yet complete.

  @4   96/10/03  Ken Moeller
       Items removed or tests added while testing @3.

  @5   96/10/04  Ken Moeller
       Removal of viewer code.  This option is no longer supported.
       Still need to investigate the resources.  So the job is not complete.

  @6   96/10/06  Ken Moeller
       Change in how units are encoded.

  @7   96/10/11  Ken Moeller
       Moved print command over to an XEvent so that the window
       can be refreshed before the screen is captured.

************************************************************ */

#include <fstream.h>                    //Added by DL 8/19/96
#include <iostream.h>                   //Added by DL 8/19/96
#include <stdlib.h>
#include <stream.h>
#include <sys/stat.h>                   //Added by DHA 9/18/96
#include <sys/types.h>                  //Added by DHA 9/18/96

#include <X11/Xatom.h>
#include <X11/cursorfont.h>
#include <X11/keysym.h>
#include <Xm/DialogS.h>                 //Added by DL 8/19/96
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/Form.h>
#include <Xm/LabelG.h>
#include <Xm/List.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PanedW.h>                  //Added by DL 8/19/96
#include <Xm/PushB.h>                   //Added by DL 8/16/96
#include <Xm/PushBG.h>                  //Added by DL 8/19/96
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectioB.h>
#include <Xm/Separator.h>               //Added by DHA 8/20/96
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/ToggleBG.h>

//#include "ge_utilities_debug.h"
#include <stdio.h>

#include "action_area.h"                //Added by DL 8/19/96
#include "build_option.h"               //Added by DHA 8/20/96
#include "ge_defs.h"                    //Added by DL 8/16/96
#include "ge_interface.h"               //Added by DL 8/16/96
#include "gettopshell.h"                //Added by DHA 8/19/96
#include "graph_editor.h"               // kbm
```

```
#include "graph_object_list.h"
//#include "op_prop_exception.h"          //Added for req. 7.4, dha 9/12/96
//#include "op_prop_formal_desc.h"        //Added for req. 7.6, dha 9/15/96
//#include "op_prop_informal_desc.h"      //Added for req. 7 dha 9/15/96
//#include "op_prop_keywords.h"           //Added for req. 7.6, dha 9/15/96
//#include "op_prop_output_guard.h"       //Added for req. 7.3, dha 9/12/96
//#include "op_prop_timer_op.h"           //Added for req. 7.5, dha 9/12/96
//#include "op_prop_timing_info.h"        //Added for req. 7 & 7.7, dha 9/5/96
//#include "op_prop_trigger_cond.h"       //Added for req. 7.2, dha 9/12/96
#include "operator_object.h"
#include "postpopup.h"                  //Added by DHA 8/20/96
#include "setcursor.h"                  //Added by DHA 8/15/96
#include "spline_object.h"
#include "stream_object.h"
#include "operator_property_menu.h"
#include "stream_property_menu.h"       //Added for req. 8, dha 9/16/96
#include "timer_tool.h"                 //Added by DL 8/22/96
#include "windows.h"                    // kbm
#include "warning.h"
#include "ge_utilities_debug.h"
#include "report_errors.h"

//   MAXCOLORS is the number of colors defined to the editor.
// To add or subtract colors, this value must be modified.

// changed size from 75
#define BUTTONWIDTH 65
#define HELPSIZ 1000

//   graph_editor has a number of global variables due to
// Motif's use of callback functions. Since these functions
// have fixed formal parameter lists, global variables must
// be used to pass some data between functions.

//   All drawing commands are executed on both drawing_a and
// drawing_area_pixmap. drawing_a is the visible canvas, while
// drawing_area_pixmap provides a backup.  When the canvas needs
// to be redrawn, the drawing in drawing_area pixmap is merely
// copied back onto the canvas.

//   colors[] is a list of predefined X colors. To use others,
// consult an X reference giving allowable color names. Using
// the predefined colors allows the user to specify color
// preferences in X resource text files.

//   graphic_list is a GraphObjectList containing all the
// visible operators and streams.

//   selected_object_ptr always points to the object selected
// (i. e. with handles around it) on the drawing canvas.

//   num_del_ops is the number of deleted operators, and
// del_op_id is an array of identifiers for deleted operators.

//   The Resrcs struct, resources[], and options[] are used
// by Motif for parsing the command line options.


  Widget toplevel, main_w, menubar, rowcol, scrolled_win,
         op_button, term_button, stream_button, select_button,
         spec_button, informal_button, types_button,
         timers_button, button_divider;
  XtAppContext app;
  Pixmap op_button_pixmap, term_button_pixmap, stream_button_pixmap,
         select_button_pixmap, spec_button_pixmap, informal_button_pixmap,
         types_button_pixmap,
         timers_button_pixmap;
  XGCValues gcv1, gcv2, gcv3;
  Screen *screen_ptr;
  XtActionsRec actions;
  String translations =
    "<Btn1Down>:   draw(down)\n\
     <Btn1Up>:     draw(up)  \n\
     <Btn1Motion>: draw(motion)\n\
     <Btn3Down>: draw(btn3down)\n\
     <Btn3Motion>: draw(btn3motion)\n\
     <Btn3Up>: draw(btn3up)\n\
     <MotionNotify>: draw(motionnotify)\n\
     <Key>:        draw(key)\n\
     <Key>Tab:    draw(tab)";
  unsigned long gc_mask;
  Window root_window, toplevel_window;
//  XEvent *print_event = (XEvent *) malloc(sizeof(XEvent));     // @7
  XEvent *print_event;

extern int    Global_argc;
extern char **Global_argv;


/*****************************************************************
*** Added by Doug Lange 8/16/96.*/
GRAPH_DESC gdnode;
ID_LIST    idp;

ACTION_NODE* next_action_ptr;          // kbm

GC std_graphics_context, dotted_context, erase_context;
Dimension  width, height;
Pixmap drawing_area_pixmap;
Widget drawing_a, current_op_name, current_op_met;
Widget save_indicator, error_indicator, status_indicator;
BOOLEAN state_stream = false, alt_selected = false, ctrl_selected = false;
BOOLEAN ibar_mode = false; // added for req #6.1. dha
BOOLEAN label_edit_mode = false; // added for req #6.1.1. dha
CLASS_DEF object_def = GRAPHOBJECT; // added for req #6.1. dha
char* colors[] = {"Aquamarine", "Black", "Blue", "BlueViolet",
```

```
                    "Brown", "CadetBlue", "Coral",
                    "CornflowerBlue", "Cyan", "DarkGreen",
                    "DarkOliveGreen", "DarkOrchid",
                    "DarkSlateBlue", "DarkSlateGrey",
                    "DarkTurquoise", "DimGrey", "Firebrick",
                    "ForestGreen", "Gold", "Goldenrod", "Grey",
                    "Green", "GreenYellow", "IndianRed", "Khaki",
                    "LightBlue", "LightGrey", "LightSteelBlue",
                    "LimeGreen", "Magenta", "Maroon",
                    "MediumAquamarine", "MediumBlue",
                    "MediumOrchid", "MediumSeaGreen",
                    "MediumSlateBlue", "MediumSpringGreen",
                    "MediumTurquoise", "MediumVioletRed",
                    "MidnightBlue", "Navy", "Orange", "OrangeRed",
                    "Orchid", "PaleGreen", "Pink", "Plum", "Red",
                    "Salmon", "SeaGreen", "Sienna", "SkyBlue",
                    "SlateBlue", "SpringGreen", "SteelBlue",
                    "Tan", "Thistle", "Turquoise", "Violet",
                    "VioletRed", "Wheat", "White", "Yellow",
                    "YellowGreen"};
unsigned long color_table[MAXCOLORS + 1];
TOOL_STATE tool_state = SELECT_TOOL;
GraphObjectList graphic_list;
GraphObject* selected_object_ptr = NULL;
OperatorObject *op_being_updated = NULL; // Add for req. 7, dha
StreamObject *st_being_updated = NULL;   // Add for req. 8, dha
Display *display_ptr;
Window draw_window;
int default_color = WHITE;
int default_font = COURIERBOLD12;
int num_del_ops = 0;
OP_ID del_op_id[MAXDELETEDOPS];
ERROR_MSGS  errors_present;
BOOLEAN psdl_modified, syntax_checked;
int     save_performed;        // updated save_state when you return
char *help_menu_files[] = {"psdl_grammar.hlp",
                           "operators.hlp",
                           "streams.hlp",
                           "exceptions.hlp",
                           "timers.hlp"};

struct _resrcs {
  int viewer;
} Resrcs;

static XtResource resources[] = {
  {"viewer", "Viewer", XmRBoolean, sizeof (int),
   XtOffsetOf(struct _resrcs,viewer), XmRImmediate, False},
};

static XrmOptionDescRec options[] = {
  {"-v", "viewer", XrmoptionNoArg, "True"},
};
```

```
void select_state(TOOL_STATE new_state) {

  tool_state = new_state;

  if (new_state == OPERATOR_TOOL)
    XtVaSetValues(op_button,       XmNshadowType, XmSHADOW_IN,  NULL);
  else
    XtVaSetValues(op_button,       XmNshadowType, XmSHADOW_OUT, NULL);

  if (new_state == TERMINATOR_TOOL)
    XtVaSetValues(term_button,     XmNshadowType, XmSHADOW_IN,  NULL);
  else
    XtVaSetValues(term_button,     XmNshadowType, XmSHADOW_OUT, NULL);

  if (new_state == STREAM_TOOL)
    XtVaSetValues(stream_button,   XmNshadowType, XmSHADOW_IN,  NULL);
  else
    XtVaSetValues(stream_button,   XmNshadowType, XmSHADOW_OUT, NULL);

  if (new_state == SELECT_TOOL)
    XtVaSetValues(select_button,   XmNshadowType, XmSHADOW_IN,  NULL);
  else
    XtVaSetValues(select_button,   XmNshadowType, XmSHADOW_OUT, NULL);
}


/*************************************************************************
 * error_label() --
 *************************************************************************/
void error_label() {

  XmString label;

  if ((errors_present == NULL) || (!syntax_checked)) {
    label = XmStringCreateSimple("Check Syntax");
    XtVaSetValues(error_indicator, XmNlabelString, label, NULL);
    XtVaSetValues(error_indicator, XmNshadowType, XmSHADOW_OUT, NULL);
  }
  else {
    label = XmStringCreateSimple("ERROR MSGS");
    XtVaSetValues(error_indicator, XmNlabelString, label, NULL);
    XtVaSetValues(error_indicator, XmNshadowType, XmSHADOW_OUT, NULL);
  }

  XmStringFree(label);
}


/*************************************************************************
 * save_state() -- Updates the save_indicator with the current indicated
 * state.
 *************************************************************************/
```

```
void save_state(int state) {

  XmString label;

  if (state == NOT_MODIFIED) {
    label = XmStringCreateSimple("Save Not Required");
    XtVaSetValues(save_indicator, XmNlabelString, label, NULL);
    XtVaSetValues(save_indicator, XmNshadowType, XmSHADOW_IN, NULL);
    psdl_modified = false;
  }
  else if (state == SAVE_REQUIRED) {
    label = XmStringCreateSimple("SAVE REQUIRED");
    XtVaSetValues(save_indicator, XmNlabelString, label, NULL);
    XtVaSetValues(save_indicator, XmNshadowType, XmSHADOW_OUT, NULL);
    psdl_modified  = true;
    syntax_checked = false;
  }
  else {
    label = XmStringCreateSimple("");
    XtVaSetValues(save_indicator, XmNlabelString, label, NULL);
    XtVaSetValues(save_indicator, XmNshadowType, XmSHADOW_IN, NULL);
  }
  XmStringFree(label);

  error_label();
}


void update_status(char *status, BOOLEAN bell) {
    XtVaSetValues(status_indicator, XmNvalue, status, NULL);
    if (bell)
      XBell(display_ptr,100);
}


void clear_status() {
    XtVaSetValues(status_indicator, XmNvalue, "", NULL);
}

//   Initializes the color table.

void initialize_color_table(Screen *screen) {
  Colormap color_map = DefaultColormapOfScreen(screen);
  XColor color, unused;
  int i, screen_depth = DefaultDepthOfScreen(screen);

  if (screen_depth > 1) {           //   a color screen
    for(i = 1; i <= MAXCOLORS; i++) {
      if (!XAllocNamedColor(display_ptr, color_map,
         colors[i - 1], &color, &unused))
   printf ("Allocated unknown color: %s\n", colors[i-1]);
      color_table[i] = color.pixel;
    }
  }
  else {                            //    a black and white screen
    for(i = 1; i <= MAXCOLORS; i++) {
      if (strcmp(colors[i - 1], "White") != 0)
        color_table[i] = BlackPixelOfScreen(screen);
      else
        color_table[i] = WhitePixelOfScreen(screen);
    }
  }
}


/*************************************************************************
*   Executes menu options from the 'file' menu. This is
*   called by either the menu callback function, if the
*   pulldown menus are used, or by the draw() function,
*   if the alt-key combinations are used.
*************************************************************************/

void handle_file_options(int item_no) {

  int  action;

  Quest_Script abort_script =
          {"","Abort changes made to graph?", "Yes", "No", "Cancel", BTN2};
  Quest_Script save_script  =
          {"","Save changes made to graph?",  "Yes", "No", "Cancel", BTN1};

  XFlush(display_ptr);
  switch(item_no) {

  case 0:  // Save

    next_action_ptr->option      = SAVE_TO_DISK;
    next_action_ptr->reinvoke    = true;
    free(next_action_ptr->next_op);
    next_action_ptr->next_op     = graphic_list.current_op_name();
    next_action_ptr->next_op_num = graphic_list.current_op_num();
    return_sde_flag              = true;

    break;

  case 1:  // Restore from Save
    action = YES;                       // Default action if not modified
    if (psdl_modified)
      action = AskUser(app,drawing_a, abort_script);
    switch(action) {
       case YES:
  next_action_ptr->option      = REVERT;
  next_action_ptr->reinvoke    = true;
  free(next_action_ptr->next_op);
  next_action_ptr->next_op     = graphic_list.current_op_name();
  next_action_ptr->next_op_num = graphic_list.current_op_num();
```

```
  return_sde_flag              = true;
  break;

       case NO:
  return_sde_flag = false;  // Aborted operation, do nothing
  break;
      }

    break;

  case 2:  // Print
    AskPrint(app,drawing_a, &PrintCmd);
    if (PrintCmd.answer == OK) {
      XSendEvent(display_ptr, toplevel_window, True, 0, print_event);
    }

    break;

  case 3:  // Exit

    action = NO;    // Default action if not modified
                    // This is not the default save option, see save_script
    if (psdl_modified)
      action = AskUser(app,drawing_a, save_script);

    switch(action) {
      case YES:
next_action_ptr->option   = SAVE_TO_DISK;
next_action_ptr->reinvoke = false;
free(next_action_ptr->next_op);
next_action_ptr->next_op     = graphic_list.root_op_name();
next_action_ptr->next_op_num = graphic_list.root_op_num();
return_sde_flag = true;
break;

      case NO:
next_action_ptr->option      = ABANDON;
next_action_ptr->reinvoke    = false;
free(next_action_ptr->next_op);
next_action_ptr->next_op     = graphic_list.root_op_name();
next_action_ptr->next_op_num = graphic_list.root_op_num();
return_sde_flag              = true;
break;

      case CANCEL:
      default:
return_sde_flag = false;
break;
    }

    break;

  default:
    break;
  }
}

/*************************************************************************
*   Executes menu options from the 'psdl' menu. This is
*   called by either the menu callback function, if the
*   pulldown menus are used, or by the draw() function,
*   if the alt-key combinations are used.
*************************************************************************/

void handle_psdl_options(int item_no) {

  int  action;
  char *opName;

  Quest_Script abort_script =
         {"","Abort changes made to graph?", "Yes", "No", "Cancel", BTN2};
  Quest_Script save_script  =
         {"","Save changes made to graph?",  "Yes", "No", "Cancel", BTN1};

  XFlush(display_ptr);
  switch(item_no) {

  case 0:  // Syntax Check

    next_action_ptr->option      = CHECK_SYNTAX;
    next_action_ptr->reinvoke    = true;
    free(next_action_ptr->next_op);
    next_action_ptr->next_op     = graphic_list.current_op_name();
    next_action_ptr->next_op_num = graphic_list.current_op_num();
    return_sde_flag              = true;

    break;

  case 1:  // Go to Root

    // Check for error condition of no Root...this should not be possible
    if (graphic_list.root_op_num() == UNDEFINED_OPNUM) {
      warning(drawing_a,"No Root node defined");

      break;
    }

    next_action_ptr->option      = UPDATE_TREE;
    next_action_ptr->reinvoke    = true;
    free(next_action_ptr->next_op);
    next_action_ptr->next_op     = graphic_list.root_op_name();
    next_action_ptr->next_op_num = graphic_list.root_op_num();
    return_sde_flag              = true;

    break;
```

```
  case 2:  // Go to Parent

    // Check for error condition of no Parent
    if (graphic_list.parent_op_num() == UNDEFINED_OPNUM) {
      warning(drawing_a,"No parent node defined");

      break;
    }

    next_action_ptr->option      = UPDATE_TREE;
    next_action_ptr->reinvoke    = true;
    free(next_action_ptr->next_op);
    next_action_ptr->next_op     = graphic_list.parent_op_name();
    next_action_ptr->next_op_num = graphic_list.parent_op_num();
    return_sde_flag = true;

    break;

  case 3:  // Decompose

    if (selected_object_ptr == NULL)
      warning(drawing_a,"Please select an operator");

    else {
      if (selected_object_ptr->is_a() == OPERATOROBJECT) {
opName = selected_object_ptr->name();
if (strchr(opName,'.') != NULL) {   // Is a type
  warning(drawing_a,"Not allowed to decompose a Type Operator");
  update_status(
      "A Type Operator must be Atomic: rename or leave Atomic",
      RING_BELL);
  free(opName);
}
else {
  next_action_ptr->option      = UPDATE_TREE;
  next_action_ptr->reinvoke    = true;
  free(next_action_ptr->next_op);
  next_action_ptr->next_op     = opName;
  next_action_ptr->next_op_num =
                    ((OperatorObject *) selected_object_ptr)->op_num();
  return_sde_flag = true;
}
      }
      else
        warning(drawing_a,"Please select an operator");
    }

    break;

  default:
    return_sde_flag = false;
    break;
  }
}


//   This function is called when a selection is made from
// the list box displayed in the 'draw_options:Color' menu.

void color_list_cb(Widget widget, XtPointer,
                   XtPointer cb_struct_ptr) {
  XmListCallbackStruct *list_struct_ptr =
    (XmListCallbackStruct *) cb_struct_ptr;

  if (selected_object_ptr != NULL) {
    if (selected_object_ptr->is_a() == OPERATOROBJECT) {
      selected_object_ptr->erase();
      selected_object_ptr->color(list_struct_ptr->item_position);
      selected_object_ptr->draw(SOLID);
      save_state(SAVE_REQUIRED);
    }
  }
  else
    default_color = list_struct_ptr->item_position;
  XtDestroyWidget(widget);
}

//   This function is called when a selection is made from
// the list box displayed in the 'draw_options:Font' menu.

void font_list_cb(Widget widget, XtPointer,
                  XtPointer cb_struct_ptr) {
  XmListCallbackStruct *list_struct_ptr =
    (XmListCallbackStruct *) cb_struct_ptr;

  if (selected_object_ptr != NULL) {
    selected_object_ptr->erase();
    selected_object_ptr->set_object_font(list_struct_ptr->item_position);
    selected_object_ptr->draw(SOLID);
    save_state(SAVE_REQUIRED);
  }
  else {
    default_font = list_struct_ptr->item_position;
    graphic_list.set_default_font(default_font);
  }
  XtDestroyWidget(widget);
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

//   This function is called when a selection is made from
// the list box displayed in the 'draw_options:Undelete Operator'
// menu.

static void op_list_cb(Widget widget, XtPointer,
```

```
                        XtPointer cb_struct_ptr) {
  XmListCallbackStruct *list_struct_ptr =
    (XmListCallbackStruct *) cb_struct_ptr;

//   The last entry in the list is 'Cancel'.
  if (list_struct_ptr->item_position != num_del_ops + 1) {

    graphic_list.set_undeleted(OPERATOROBJECT,
                               del_op_id[list_struct_ptr->item_position - 1]);
    save_state(SAVE_REQUIRED);
    graphic_list.draw();
  }
  XtDestroyWidget(widget);
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

//   Executes menu options from the 'Edit' menu. This is
// called by either the menu callback function, if the
// pulldown menus are used, or by the draw() function,
// if the alt-key combinations are used.

void handle_edit_options(int item_no) {
  int i, num_items = XtNumber(colors);
  int reply;
  XmStringTable color_list, font_list, op_list;
  Widget list_box, op_box;
  char *del_op_str[MAXDELETEDOPS];

  switch(item_no) {
    case 0:
      color_list =
        (XmStringTable) XtMalloc(num_items * sizeof(XmString *));
      for(i = 0; i < num_items; i++)
        color_list[i] = XmStringCreateSimple(colors[i]);
      list_box =
              XmCreateScrolledList(drawing_a, "Colors", NULL, 0);
      XtVaSetValues(list_box,
        XmNitems, color_list,
        XmNitemCount, num_items,
        XmNvisibleItemCount, 8,
        NULL);
      for(i = 0; i < num_items; i++)
        XmStringFree(color_list[i]);
      XtFree((char *) color_list);
      XtAddCallback(list_box, XmNdefaultActionCallback,
                    color_list_cb, NULL);
      XtManageChild(list_box);
      break;

    case 1:
      font_list =
         (XmStringTable) XtMalloc(MAXFONTS * sizeof(XmString *));
      for(i = 0; i < MAXFONTS; i++)
        font_list[i] =
             XmStringCreateSimple(graphic_list.font_name(i + 1));
      list_box =
              XmCreateScrolledList(drawing_a, "Fonts", NULL, 0);
      XtVaSetValues(list_box,
        XmNitems, font_list,
        XmNitemCount, MAXFONTS,
        XmNvisibleItemCount, 7,
        NULL);
      for(i = 0; i < MAXFONTS; i++)
        XmStringFree(font_list[i]);
      XtFree((char *) font_list);
      XtAddCallback(list_box, XmNdefaultActionCallback,
                    font_list_cb, NULL);
      XtManageChild(list_box);
      break;

    case 2:
      if (selected_object_ptr != NULL) {
        selected_object_ptr->unselect();
        selected_object_ptr = NULL;
      }
      graphic_list.get_del_op_list(del_op_str, del_op_id,
                                   num_del_ops);
      op_list = (XmStringTable)
                XtMalloc((num_del_ops + 1) * sizeof(XmString *));
      for(i = 0; i < num_del_ops; i++)
        op_list[i] = XmStringCreateSimple(del_op_str[i]);
      op_list[num_del_ops] = XmStringCreateSimple("Cancel");
      op_box = XmCreateScrolledList(drawing_a, "Undelete",
                                    NULL, 0);
      XtVaSetValues(op_box,
                    XmNitems, op_list,
                    XmNitemCount, num_del_ops + 1,
                    XmNvisibleItemCount, 7,
                    NULL);
      for(i = 0; i < num_del_ops + 1; i++)
        XmStringFree(op_list[i]);
      XtFree((char *) op_list);
      XtAddCallback(op_box, XmNdefaultActionCallback,
                    op_list_cb, NULL);
      XtManageChild(op_box);
      break;

    case 3:
      Quest_Script abandon_script  =
        {"","All changes will be lost, are you sure?",
 "Yes","No","Cancel", BTN1};

      reply = AskUser(app, drawing_a, abandon_script);
      if (reply == YES) {
next_action_ptr->option      = ABANDON;
next_action_ptr->reinvoke    = true;
```

```
free(next_action_ptr->next_op);
next_action_ptr->next_op     = graphic_list.current_op_name();
next_action_ptr->next_op_num = graphic_list.current_op_num();
return_sde_flag              = true;
      }

    break;

    case 4:
      XFillRectangle(display_ptr, drawing_area_pixmap,
     erase_context, 0, 0, width, height);
      XFillRectangle(display_ptr, draw_window,
     erase_context, 0, 0, width, height);
      graphic_list.draw();

      break;

    default:
      break;
  }
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

void handle_tool_options(int item_no) {

  warning(drawing_a,"Not yet implemented.");
}

void set_color(Widget widget, char *color) {
  Display *dpy = XtDisplay(widget);
  Colormap cmap = DefaultColormapOfScreen(XtScreen(widget));
  XColor col, unused;

  if (!XAllocNamedColor(dpy, cmap, color, &col, &unused)) {
    warning(drawing_a,"Can't allocate color");
    return;
  }
  XSetForeground(dpy, std_graphics_context, col.pixel);
}


/*********************************************************************
 * Menu call-back functions.  These functions are called by the window
 * manager when a menu option is selected from a pull-down menu.  The
 * item which was selected is passed in client_data.
 *********************************************************************/

static void file_menu_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_file_options(item_no);
}
```

```
static void psdl_menu_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_psdl_options(item_no);
}

static void edit_menu_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_edit_options(item_no);
}

static void tool_menu_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_tool_options(item_no);
}

static void help_menu_cb(Widget w, XtPointer client_data,
 XtPointer call_data) {
  int item_no = (int) client_data;

  help_cb(drawing_a, help_menu_files[item_no], call_data);

}

void help_cb(Widget w, XtPointer client_data, XtPointer call_data) {
// Implemented by Doug Lange 8/19/96

  Widget help_dialog, pane, text_w, rc, action_a;
  struct stat statb;

  char ch, *buf;
  int i = 0, n = 0;
  int len = 0;
  Arg     args[10];

  static ActionAreaItem   action_items[] = {
    {"OK", close_dialog, NULL}
  };

  help_dialog = XtVaCreatePopupShell("Help",
     xmDialogShellWidgetClass, XtParent(w),
     XmNdeleteResponse, XmDESTROY,
     NULL);

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, help_dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass, pane, NULL);
```

```
  stat((char*)client_data, &statb);
  ifstream from((char *)client_data);
  len = statb.st_size;
  buf = new char[len +1]; // Add a space for NULL
  i = 0;
  //      while (from.get(ch) && (i < HELPSIZ -1)) {
  while (from.get(ch) && (i < len)) {
    buf[i] = ch;
    i++;
  }
  buf[i] = (char)NULL;

  XtSetArg(args[n], XmNscrollVertical,       true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,     false); n++;
  XtSetArg(args[n], XmNeditMode,             XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,             false); n++;
  XtSetArg(args[n], XmNcursorPositionVisible,false); n++;
  XtSetArg(args[n], XmNwordWrap,             true); n++;
  XtSetArg(args[n], XmNvalue,                buf); n++;
  XtSetArg(args[n], XmNrows,                 20); n++;
  XtSetArg(args[n], XmNwidth,                525); n++;
  text_w = XmCreateScrolledText(rc, "help_text", args, n);

  delete buf;

  XtManageChild(text_w);
  XtManageChild(rc);

  action_items[0].data = (XtPointer)help_dialog;
  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  XtManageChild(pane);
  XtPopup(help_dialog, XtGrabNone);

}


void build_menu_bar(Widget &main_w, Widget &menubar) {
  // 8/4/96 KBM  Updated for label changes in Req 4
  //             Also changed callback names to reflect new labels.

  XmString
    file_menu,  save_opt, restore_opt, print_opt, exit_opt,
    psdl_menu,  syntax_check_opt, goto_root_opt, goto_parent_opt,
                decompose_opt,
    edit_menu,  color_opt, font_opt, undelete_opt, abandon_opt, refresh_opt,
//  tool_menu,  reuse_lib_opt,
    help_menu,  psdl_grammar_opt, operator_opt, stream_opt, exception_opt,
                timer_opt;

  Widget widget;

  file_menu         = XmStringCreateSimple("File");
    save_opt        = XmStringCreateSimple("Save");
    restore_opt     = XmStringCreateSimple("Restore from Save");
    print_opt       = XmStringCreateSimple("Print");
    exit_opt        = XmStringCreateSimple("Exit");

  psdl_menu         = XmStringCreateSimple("PSDL");
    syntax_check_opt= XmStringCreateSimple("Syntax Check");
    goto_root_opt   = XmStringCreateSimple("Go to Root");
    goto_parent_opt = XmStringCreateSimple("Go to Parent");
    decompose_opt   = XmStringCreateSimple("Decompose");

  edit_menu         = XmStringCreateSimple("Edit");
    color_opt       = XmStringCreateSimple("Color");
    font_opt        = XmStringCreateSimple("Font");
    undelete_opt    = XmStringCreateSimple("Undelete Operator");
    abandon_opt     = XmStringCreateSimple("Abandon Changes");
    refresh_opt     = XmStringCreateSimple("Refresh Display");
  /*
  tool_menu         = XmStringCreateSimple("Tools");
    reuse_lib_opt   = XmStringCreateSimple("Reuse Library");
  */
  help_menu         = XmStringCreateSimple("Help");
    psdl_grammar_opt= XmStringCreateSimple("PSDL Grammar");
    operator_opt    = XmStringCreateSimple("Operators");
    stream_opt      = XmStringCreateSimple("Streams");
    exception_opt   = XmStringCreateSimple("Exceptions");
    timer_opt       = XmStringCreateSimple("Timers");

  menubar = XmVaCreateSimpleMenuBar(main_w, "menubar",
      XmVaCASCADEBUTTON, file_menu, NULL,
      XmVaCASCADEBUTTON, psdl_menu, NULL,
      XmVaCASCADEBUTTON, edit_menu, NULL,
//    XmVaCASCADEBUTTON, tool_menu, NULL,
      XmVaCASCADEBUTTON, help_menu, NULL, NULL);

  if (widget = XtNameToWidget(menubar, "button_3"))          // Assign to help
    XtVaSetValues(menubar, XmNmenuHelpWidget, widget, NULL);

  XmVaCreateSimplePulldownMenu(menubar, "file_menu", 0, file_menu_cb,
    XmVaPUSHBUTTON, save_opt,          NULL, NULL, NULL,
    XmVaPUSHBUTTON, restore_opt,       NULL, NULL, NULL,
    XmVaPUSHBUTTON, print_opt,         NULL, NULL, NULL,
    XmVaPUSHBUTTON, exit_opt,          NULL, NULL, NULL,
    NULL);

  XmVaCreateSimplePulldownMenu(menubar, "psdl_menu", 1, psdl_menu_cb,
    XmVaPUSHBUTTON, syntax_check_opt, NULL, NULL, NULL,
    XmVaPUSHBUTTON, goto_root_opt,     'R', NULL, NULL,
    XmVaPUSHBUTTON, goto_parent_opt,   'P', NULL, NULL,
    XmVaPUSHBUTTON, decompose_opt,     'D', NULL, NULL,
    NULL);

  XmVaCreateSimplePulldownMenu(menubar, "edit_menu", 2, edit_menu_cb,
```

```
    XmVaPUSHBUTTON, color_opt,        NULL, NULL, NULL,
    XmVaPUSHBUTTON, font_opt,         NULL, NULL, NULL,
    XmVaPUSHBUTTON, undelete_opt,     NULL, NULL, NULL,
    XmVaPUSHBUTTON, abandon_opt,      NULL, NULL, NULL,
    XmVaPUSHBUTTON, refresh_opt,       'f', NULL, NULL,
    NULL);

  /*
  XmVaCreateSimplePulldownMenu(menubar, "tool_menu", 3, tool_menu_cb,
    XmVaPUSHBUTTON, reuse_lib_opt,   'U', NULL, NULL,
    NULL);
  */

  XmVaCreateSimplePulldownMenu(menubar, "help_menu", 3, help_menu_cb,
    XmVaPUSHBUTTON, psdl_grammar_opt, NULL, NULL, NULL,
    XmVaPUSHBUTTON, operator_opt,     NULL, NULL, NULL,
    XmVaPUSHBUTTON, stream_opt,       NULL, NULL, NULL,
    XmVaPUSHBUTTON, exception_opt,    NULL, NULL, NULL,
    XmVaPUSHBUTTON, timer_opt,        NULL, NULL, NULL,
    NULL);

  XmStringFree(file_menu);
    XmStringFree(save_opt);
    XmStringFree(restore_opt);
    XmStringFree(print_opt);
    XmStringFree(exit_opt);
  XmStringFree(psdl_menu);
    XmStringFree(syntax_check_opt);
    XmStringFree(goto_root_opt);
    XmStringFree(goto_parent_opt);
    XmStringFree(decompose_opt);
  XmStringFree(edit_menu);
    XmStringFree(color_opt);
    XmStringFree(font_opt);
    XmStringFree(undelete_opt);
    XmStringFree(abandon_opt);
    XmStringFree(refresh_opt);
//XmStringFree(tool_menu);
//  XmStringFree(reuse_lib_opt);
  XmStringFree(help_menu);
    XmStringFree(psdl_grammar_opt);
    XmStringFree(operator_opt);
    XmStringFree(stream_opt);
    XmStringFree(exception_opt);
    XmStringFree(timer_opt);
}

//   Creates the push buttons used to select the tools.

void make_buttons(Widget  &rowcol,
  Widget  &op_button,           // tools
                Widget  &term_button,
  Widget  &stream_button,
                Widget  &select_button,
  Widget  &types_button,        // types
  Widget  &spec_button,         // current op spec
                Widget  &timers_button,       // current op impl
  Widget  &informal_button,     // current op impl

                Pixmap  &op_button_pixmap,
                Pixmap  &term_button_pixmap,
                Pixmap  &stream_button_pixmap,
                Pixmap  &select_button_pixmap,
                Pixmap  &types_button_pixmap,
  Pixmap  &spec_button_pixmap,
                Pixmap  &timers_button_pixmap,
  Pixmap  &informal_button_pixmap,
                Display *display_ptr,
  Screen  *screen_ptr) {

  static Widget op_btn_bb, term_btn_bb, stream_btn_bb, select_btn_bb,
               types_btn_bb, spec_btn_bb, timers_btn_bb, informal_btn_bb;

  Window root_window = RootWindowOfScreen(screen_ptr);
  unsigned int screen_depth = DefaultDepthOfScreen(screen_ptr);


  op_button_pixmap       = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);
  term_button_pixmap     = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);
  stream_button_pixmap   = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);
  select_button_pixmap   = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);
  types_button_pixmap    = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);
  spec_button_pixmap     = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);
  timers_button_pixmap   = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);
  informal_button_pixmap = XCreatePixmap(display_ptr, root_window,
   BUTTONWIDTH-4, BUTTONWIDTH-4, screen_depth);

  XFillRectangle(display_ptr, (Drawable) op_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);
  XFillRectangle(display_ptr, (Drawable) term_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);
  XFillRectangle(display_ptr, (Drawable) stream_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);
  XFillRectangle(display_ptr, (Drawable) select_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);
  XFillRectangle(display_ptr, (Drawable) types_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);
  XFillRectangle(display_ptr, (Drawable) spec_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);
```

```
 XFillRectangle(display_ptr, (Drawable) timers_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);
 XFillRectangle(display_ptr, (Drawable) informal_button_pixmap,
erase_context, 0, 0, BUTTONWIDTH-4, BUTTONWIDTH-4);

 XSetLineAttributes(display_ptr, std_graphics_context, 2,
                    LineSolid, CapButt, JoinMiter);

 XDrawArc(display_ptr, (Drawable) op_button_pixmap,
          std_graphics_context,
  10, 15, BUTTONWIDTH-(3*10), BUTTONWIDTH-(3*10),
  CIRCLE_BEGIN, FULL_CIRCLE);
 XDrawRectangle(display_ptr, (Drawable) term_button_pixmap,
                std_graphics_context,
10, 15, BUTTONWIDTH-(3*10), BUTTONWIDTH-(3*10));
 XDrawLine(display_ptr, (Drawable) stream_button_pixmap,
           std_graphics_context,
   10, 15, BUTTONWIDTH-(2*10), BUTTONWIDTH-(2*10));
 XDrawString(display_ptr, (Drawable) select_button_pixmap,
             std_graphics_context, 10, (BUTTONWIDTH/2)+5, "Select", 6);
 XDrawString(display_ptr, (Drawable) types_button_pixmap,
             std_graphics_context, 10, (BUTTONWIDTH/2)+5, "Types ", 6);
 XDrawString(display_ptr, (Drawable) spec_button_pixmap,
             std_graphics_context, 10, (BUTTONWIDTH/2)+5, " Spec ", 6);
 XDrawString(display_ptr, (Drawable) timers_button_pixmap,
             std_graphics_context, 10, (BUTTONWIDTH/2)+5, "Timers", 6);
 XDrawString(display_ptr, (Drawable) informal_button_pixmap,
             std_graphics_context, 10, (BUTTONWIDTH/2)-8, "Graph ", 6);
 XDrawString(display_ptr, (Drawable) informal_button_pixmap,
             std_graphics_context, 5, (BUTTONWIDTH/2)+5, "Informal", 8);
 XDrawString(display_ptr, (Drawable) informal_button_pixmap,
             std_graphics_context, 10, (BUTTONWIDTH/2)+18, "Desc  ", 6);

 XmString button_label;

 button_label = XmStringCreateSimple("Operator");

 op_button = XtVaCreateManagedWidget("op_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);

 XmStringFree(button_label);
 button_label = XmStringCreateSimple("  Term");

 term_button = XtVaCreateManagedWidget("term_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);

 XmStringFree(button_label);
 button_label = XmStringCreateSimple(" Stream");

 stream_button = XtVaCreateManagedWidget("stream_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);

 XmStringFree(button_label);
 button_label = XmStringCreateSimple(" Select");

 select_button = XtVaCreateManagedWidget("select_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);

 XmStringFree(button_label);

 button_divider = XtVaCreateManagedWidget("separator",
                        xmSeparatorWidgetClass, rowcol,
```

```
//   XmNy,     ROW14 - 8,
//   XmNwidth, WIN_WIDTH,
  NULL);


  button_label = XmStringCreateSimple("  Types");

  types_button = XtVaCreateManagedWidget("types_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);

  XmStringFree(button_label);
  button_label = XmStringCreateSimple("  Spec");

  spec_button = XtVaCreateManagedWidget("spec_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);

  XmStringFree(button_label);
  button_label = XmStringCreateSimple(" Timers");

  timers_button = XtVaCreateManagedWidget("timers_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);

  XmStringFree(button_label);
  button_label = XmStringCreateLtoR("  Graph\n  Desc",
    XmSTRING_DEFAULT_CHARSET);

  informal_button = XtVaCreateManagedWidget("informal_button",
xmDrawnButtonWidgetClass,
rowcol,
XmNrecomputeSize, false,
XmNpushButtonEnabled, false,
XmNshadowType, XmSHADOW_OUT,
XmNwidth, BUTTONWIDTH,
XmNheight, BUTTONWIDTH,
XmNlabelType, XmSTRING,
XmNlabelString, button_label,
//XmNlabelType, XmPIXMAP,
//XmNlabelPixmap, op_button_pixmap,
NULL);
  XmStringFree(button_label);

  XSetLineAttributes(display_ptr, std_graphics_context, 1,
                     LineSolid, CapButt, JoinMiter);

}

//   Redraws the drawing canvas.

void redraw(Widget, XtPointer,
            XtPointer cbs) {
  XmDrawingAreaCallbackStruct *temp_ptr;

  temp_ptr = (XmDrawingAreaCallbackStruct *) cbs;
  XCopyArea(temp_ptr->event->xexpose.display,
            drawing_area_pixmap, temp_ptr->window,
            std_graphics_context, 0, 0, width, height, 0, 0);
}

//   Draws a square black box on the canvas to aid in
// graphic manipulation of objects.

void draw_handle(CC graphics_context, int x, int y) {

  x -= HANDLESIZE / 2;
  y -= HANDLESIZE / 2;
  if (x < 0)
    x = 0;
  if (y < 0)
    y = 0;

//  When the display function is set to GXxor, the pixel being
// written is exclusive-or'ed with the target pixel to
// determine color.  This means that writing the same pixel with
```

```
// the same color twice restores the original color, simplifying
// the process of erasing handles.

  XSetFunction(display_ptr, graphics_context, GXxor);
  XFillRectangle(display_ptr, draw_window, graphics_context,
                 x, y, HANDLESIZE, HANDLESIZE);
  XFillRectangle(display_ptr, drawing_area_pixmap,
                 graphics_context,
                 x, y, HANDLESIZE, HANDLESIZE);
  XSetFunction(display_ptr, graphics_context, GXcopy);
}


//   This function erases the temporary guidelines used when
// streams are drawn.
//   Dotted lines are erased first, then handles.  Since each
// handle is overwritten with the following dotted line, an
// erased handle makes an erased blotch in the beginning of the
// next segment.   When the next segment is written in xor mode,
// it makes a black mark where the erased handle overwrote the
// beginning of its segment.

void erase_guides(OP_ID from_stream_id, SplineObject *temp_spline_ptr) {
  OperatorObject *temp_operator_ptr;
  XYPAIR line_start, line_end;

  temp_spline_ptr->reset_iter();
  if (from_stream_id != UNDEFINED_OPNUM) {
    temp_operator_ptr = (OperatorObject *)
      graphic_list.target_object(OPERATOROBJECT, from_stream_id);
    line_start = temp_operator_ptr->center();
  }
  else
    line_start = temp_spline_ptr->next_pair();
  line_end = temp_spline_ptr->next_pair();
  while(line_end.x != -1) {
    XDrawLine(display_ptr, draw_window, dotted_context,
              line_start.x, line_start.y, line_end.x,
              line_end.y);
    XDrawLine(display_ptr,drawing_area_pixmap, dotted_context,
              line_start.x, line_start.y, line_end.x,
              line_end.y);
    line_start = line_end;
    line_end = temp_spline_ptr->next_pair();
  }
  temp_spline_ptr->reset_iter();
  line_end = temp_spline_ptr->next_pair();
  while(line_end.x != -1) {
    draw_handle(std_graphics_context, line_end.x, line_end.y);
    line_end = temp_spline_ptr->next_pair();
  }
}
```

```
//   This function is called when a stream is being drawn
// and the mouse is clicked on either a clear spot on the
// drawing canvas, or on top of another stream.  If a double-
// click is registered, the user wants to terminate an external
// stream.

void handle_null_point(OP_ID from_stream_id, int &last_point_x,
       int &last_point_y,
       int &x_state, int &y_state,
       XEvent in_event,
       SplineObject *temp_spline_ptr, BOOLEAN &done,
       GraphObject *&temp_object_ptr,
       StreamObject *&temp_stream_ptr) {

//   Checks for two clicks in the same spot.

  if ((from_stream_id != UNDEFINED_OPNUM) &&
     ((last_point_x - (HANDLESIZE / 2) - HITFUDGE)
      < in_event.xbutton.x) &&
     ((last_point_x + (HANDLESIZE / 2) + HITFUDGE)
      > in_event.xbutton.x) &&
     ((last_point_y - (HANDLESIZE / 2) - HITFUDGE)
      < in_event.xbutton.y) &&
     ((last_point_y + (HANDLESIZE / 2) + HITFUDGE)
      > in_event.xbutton.y)) {
    erase_guides(from_stream_id, temp_spline_ptr);
    OP_ID new_id = graphic_list.request_id(STREAMOBJECT);
    temp_stream_ptr = new StreamObject("", new_id,
                                       from_stream_id,
                                       0, UNDEFINED_TIME, MS,
       temp_spline_ptr,// @6
       true, false);
    temp_stream_ptr->set_object_ptrs(&graphic_list);
    graphic_list.add(temp_stream_ptr);
    save_state(SAVE_REQUIRED);
    temp_stream_ptr->draw(SOLID);
    temp_stream_ptr = NULL;
    temp_object_ptr = NULL;
    done = true;
    temp_spline_ptr->clear();
  }
  else {
    x_state = in_event.xbutton.x;
    y_state = in_event.xbutton.y;
    temp_spline_ptr->add(x_state, y_state);
    XDrawLine(display_ptr, draw_window, dotted_context,
              last_point_x, last_point_y, x_state, y_state);
    XDrawLine(display_ptr, drawing_area_pixmap, dotted_context,
              last_point_x, last_point_y, x_state, y_state);
    draw_handle(std_graphics_context, x_state, y_state);
#ifdef GE_DEBUG
//    cout << "ge: " << x_state << " " << y_state << " " <<
```

```
//      HANDLESIZE << endl;
#endif
    last_point_x = x_state;
    last_point_y = y_state;
  }
}


//   Once the user selects the Stream Tool and begins to draw,
// the draw_stream() function handles all events to speed up
// performance.

void draw_stream(int initial_x, int initial_y) {
  GraphObject *temp_object_ptr;
  OperatorObject *conv_ptr;
  XYPAIR temp_pair;
  char buffer[INPUT_LINE_SIZE]; // added for req #6.4 dha
  int count = 0; // added for req #6.4 dha
  int bufsize = INPUT_LINE_SIZE; // added for req #6.4 dha
  OP_ID from_stream_id;
  int x_state, y_state, last_point_x, last_point_y;
  unsigned long stream_event_mask =
                              (ButtonPressMask | PointerMotionMask |
                               KeyPressMask);
  unsigned long normal_mask = (ButtonPressMask | PointerMotionMask |
                               KeyPressMask |
                               ButtonMotionMask | ExposureMask |
                               ButtonReleaseMask);
  XEvent in_event;
  StreamObject *temp_stream_ptr;
  SplineObject *temp_spline_ptr;
  BOOLEAN done = false;
  KeySym keysym; // added for req #6.4 dha

  temp_spline_ptr = new SplineObject;

  temp_object_ptr = graphic_list.hit(initial_x, initial_y);
  if (temp_object_ptr == NULL) {  // External stream
    from_stream_id = UNDEFINED_OPNUM;
    temp_spline_ptr->add(initial_x, initial_y);
    x_state = initial_x;
    y_state = initial_y;
    draw_handle(std_graphics_context, x_state, y_state);
  }
  else {
    if (temp_object_ptr->is_a() != OPERATOROBJECT) {
  //  External Stream
      from_stream_id = UNDEFINED_OPNUM;
      temp_spline_ptr->add(initial_x, initial_y);
      x_state = initial_x;
      y_state = initial_y;
      draw_handle(std_graphics_context, x_state, y_state);
    }
    else {
      conv_ptr = (OperatorObject *) temp_object_ptr;
      from_stream_id = conv_ptr->id();
      temp_object_ptr = NULL;
      temp_pair = conv_ptr->center();
      x_state = temp_pair.x;
      y_state = temp_pair.y;
    }
  }
  last_point_x = x_state;
  last_point_y = y_state;
  XSelectInput(display_ptr, draw_window,
               stream_event_mask);
  while(done == false) {  // monitors the event loop
    XNextEvent(display_ptr, &in_event);
    if (in_event.xbutton.window == draw_window) {
      switch(in_event.type) {
        case MotionNotify:

#ifdef GE_DEBUG
  //          cout << "Motion" << endl;
#endif

          XDrawLine(display_ptr, draw_window, dotted_context,
                  last_point_x, last_point_y, x_state, y_state);
          XDrawLine(display_ptr, drawing_area_pixmap,
                    dotted_context, last_point_x, last_point_y,
                    x_state, y_state);
          x_state = in_event.xbutton.x;
          y_state = in_event.xbutton.y;
          XDrawLine(display_ptr, draw_window, dotted_context,
                  last_point_x, last_point_y, x_state, y_state);
          XDrawLine(display_ptr, drawing_area_pixmap,
                    dotted_context, last_point_x, last_point_y,
                    x_state, y_state);
          break;
        case ButtonPress:
        case KeyPress:
#ifdef GE_DEBUG
          if (in_event.type == ButtonPress) {
//            cout << "buttonpress" << endl;
          }
          else {
//            cout << "keypress" << endl;
          }
#endif
          XDrawLine(display_ptr, draw_window, dotted_context,
                   last_point_x, last_point_y, x_state, y_state);
          XDrawLine(display_ptr, drawing_area_pixmap,
                    dotted_context, last_point_x, last_point_y,
                    x_state, y_state);
          temp_object_ptr = graphic_list.hit(in_event.xbutton.x,
                                             in_event.xbutton.y);
```

```
      if (temp_object_ptr == NULL) {
        handle_null_point(from_stream_id, last_point_x,
                          last_point_y, x_state, y_state,
                          in_event, temp_spline_ptr, done,
                          temp_object_ptr, temp_stream_ptr);
      }
      else
        if (temp_object_ptr->is_a() == OPERATOROBJECT) {
          erase_guides(from_stream_id, temp_spline_ptr);
          OP_ID new_id = graphic_list.request_id(STREAMOBJECT);
          temp_stream_ptr =
               new StreamObject("", new_id, from_stream_id,
                                temp_object_ptr->id(),
UNDEFINED_TIME, MS,   // @6
                                temp_spline_ptr, true, false);
          temp_stream_ptr->set_object_ptrs(&graphic_list);
  save_state(SAVE_REQUIRED);
          graphic_list.add(temp_stream_ptr);
          temp_stream_ptr->draw(SOLID);
          temp_stream_ptr = NULL;
          temp_object_ptr = NULL;
          done = true;
          temp_spline_ptr->clear();
      }
      else
        if (temp_object_ptr->is_a() == STREAMOBJECT) {
          handle_null_point(from_stream_id, last_point_x,
                            last_point_y, x_state, y_state,
                            in_event, temp_spline_ptr, done,
                            temp_object_ptr, temp_stream_ptr);
      }

      if (in_event.type == KeyPress) {
        count = XLookupString(&in_event.xkey, buffer,
          bufsize, &keysym, NULL);
        buffer[count] = NULL; /* add NULL terminator */

        if (keysym == XK_Escape) {

          //temp_stream_ptr->erase();
          //OP_ID deleted_op_id = temp_stream_ptr->id();
          //graphic_list.delete_notify(temp_stream_ptr->
          //                           is_a(), deleted_op_id);
          //temp_stream_ptr->set_deleted();
          //temp_stream_ptr = NULL;
          graphic_list.draw();

          done = true;
        }
        else {
          XBell(display_ptr, 100);
        }
      }
```



```
          break;

        default:
          ;
          break;
      }  //switch
    } //if right window
  }  //while done == false
  done = false;
  XSelectInput(display_ptr, draw_window, normal_mask);
}


//   Draws the outline of the text being moved.

void draw_text_shadow(int x, int y, int width, int height) {

  XDrawRectangle(display_ptr, drawing_area_pixmap,
                 dotted_context, x - width / 2, y - height / 2,
                 width, height);
  XDrawRectangle(display_ptr, draw_window, dotted_context,
                 x - width / 2, y - height / 2, width, height);
}


//   The main draw routine.  This function is called by the
// window manager every time the mouse is moved, a mouse button
// pressed, or a key pressed inside the draw window.  It is
// called with a string token that indicates why it was called,
// and processes the event accordingly.

void draw(Widget, XEvent *event, String *args, Cardinal *) { // void draw
  static char string[INPUT_LINE_SIZE]; // added for req #6.1.1 dha
  static OperatorObject *temp_operator_ptr = NULL;
  static StreamObject *temp_stream_ptr = NULL;
  static BOOLEAN first_draw = true, handle_selected = false,
                 text_selected = false, drawing_changed = false;
  static int x_state, y_state, shadow_height, shadow_width;
  static OP_ID from_stream_id;
  GraphObject *temp_object_ptr = NULL;
  static GraphObject *ibar_object_ptr = NULL;
  char   *warningMSG;
  char buffer[INPUT_LINE_SIZE]; // added for req #6.1.1 dha
  int count = 0; // added for req #6.1.1 dha
  int length = 0; // added for req #6.1.1 dha
  int bufsize = INPUT_LINE_SIZE; // added for req #6.1.1 dha
  int x = event->xbutton.x;
  int y = event->xbutton.y;
  char *labelName;
  OperatorObject *conv_op_ptr;
  StreamObject *conv_st_ptr;
  KeySym keysym; // added for req #6.1.1 dha
  BOOLEAN state_change, type_match;
```

```
BOOLEAN type_operator;

if (strcmp(args[0], "down") == 0) {  // Button pressed
  clear_status();
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  x_state = x;
  y_state = y;
  if (tool_state == SELECT_TOOL) {
    if (selected_object_ptr != NULL) {
      if (selected_object_ptr->hit_handle(x, y)) {
        handle_selected = true;
        object_def = selected_object_ptr->is_a();
        if (object_def == OPERATOROBJECT) {
          op_being_updated = (OperatorObject *)selected_object_ptr;
          delete temp_operator_ptr;
          conv_op_ptr = (OperatorObject *) selected_object_ptr;
          temp_operator_ptr =
            new OperatorObject("", UNDEFINED_OPNUM, UNDEFINED_OPNUM,
UNDEFINED_TIME, MS, conv_op_ptr->x(),  // @6
                               conv_op_ptr->y(),
                               conv_op_ptr->radius(),
                               default_color, false,
                               conv_op_ptr->is_composite(),
                               conv_op_ptr->is_terminator());
          temp_operator_ptr->set_handle_selected(
       conv_op_ptr->handle_selected());
        } else if (object_def == STREAMOBJECT) {
          st_being_updated = (StreamObject *)selected_object_ptr;
        }
      }  // selected_object_ptr->hit_handle()
      else {          // Unselects previously selected object
        handle_selected = false;
        selected_object_ptr->unselect();
        selected_object_ptr = NULL;
        delete temp_operator_ptr;
        temp_operator_ptr = NULL;
      }
    }  // selected_object_ptr != NULL

    if (handle_selected == false) {
      temp_object_ptr = graphic_list.hit(x, y);
      if (temp_object_ptr != NULL) {
        temp_object_ptr->select();
        selected_object_ptr = temp_object_ptr;
        text_selected = selected_object_ptr->text_selected();
        object_def = temp_object_ptr->is_a();
        if (object_def == OPERATOROBJECT) {
          op_being_updated = (OperatorObject *) temp_object_ptr;

  // Makes temporary operator to move around
          delete temp_operator_ptr;
          conv_op_ptr = (OperatorObject *) temp_object_ptr;
          temp_operator_ptr =
              new OperatorObject("", UNDEFINED_OPNUM, UNDEFINED_OPNUM,
 UNDEFINED_TIME, MS, conv_op_ptr->x(),    // @6
                                 conv_op_ptr->y(),
                                 conv_op_ptr->radius(),
                                 default_color, false,
                                 conv_op_ptr->is_composite(),
                                 conv_op_ptr->is_terminator());
        } else if (object_def == STREAMOBJECT) {
          st_being_updated = (StreamObject *) temp_object_ptr;
        }
      } // temp_object_ptr != NULL
      else { // No object selected
        temp_object_ptr = NULL;
        selected_object_ptr = NULL;
        delete temp_operator_ptr;
        temp_operator_ptr = NULL;
      }
    } // handle_selected == false
  } /// tool_state == SELECT_TOOL
  else {  // button down, operator tool selected?
    //      if ((((tool_state == OPERATOR_TOOL) ||
    //           (tool_state == TERMINATOR_TOOL)) &&
    //          (ibar_mode != true)) ||                 // added 8/22/96 dha,
    //         (object_def != OPERATOROBJECT)) {        //  req. 6.2 & 6.3
    if ((tool_state == OPERATOR_TOOL) ||
 (tool_state == TERMINATOR_TOOL)) {
       OP_ID new_id = graphic_list.request_id(OPERATOROBJECT);
OP_ID new_op = graphic_list.request_id(OPERATOROBJECT);
       if (tool_state == OPERATOR_TOOL) {
         temp_operator_ptr =
   // BROCKETT 1/22/93 default x and y values changed from 0 to 100
           new OperatorObject("", new_id, new_op,
      UNDEFINED_TIME, MS, 100, 100, 30,   // @6
                          default_color, true, false,
                          false);
         temp_operator_ptr->set_location(x, y);
       } // tool_sate == OPERATOR_TOOL
       else
         if (tool_state == TERMINATOR_TOOL) {
           temp_operator_ptr =
     // BROCKETT 1/22/93 default x and y values changed from 0 to 100
             new OperatorObject("", new_id, new_op,
 UNDEFINED_TIME, MS, 100, 100, 30,  // @6
                             default_color, true, false,
                             true);
           temp_operator_ptr->set_location(x, y);
         } // tool_state == TERMINATOR_TOOL
       graphic_list.add((GraphObject *) temp_operator_ptr);
save_state(SAVE_REQUIRED);
       temp_operator_ptr->draw(SOLID);
       temp_operator_ptr = NULL;
     }  // tool_state == OPERATOR_TOOL || TERMINATOR_TOOL && ibar_mode ||
     else // button down, stream tool selected?
```

```
        if (tool_state == STREAM_TOOL) {
          draw_stream(x, y);
        }
    }  // button down, operator tool selected?
  } else if (strcmp(args[0], "motion") == 0) {  // button not down
    if (tool_state == SELECT_TOOL) {
      if (selected_object_ptr != NULL) {
drawing_changed = true;
if (text_selected) {
  if (first_draw == true) {
    shadow_width = selected_object_ptr->text_width();
    shadow_height = selected_object_ptr->text_height();
    draw_text_shadow(x, y, shadow_width, shadow_height);
    first_draw = false;
  } // first_draw
  else {
    draw_text_shadow(x_state, y_state, shadow_width, shadow_height);
    draw_text_shadow(x, y, shadow_width, shadow_height);
  }
} // text_selected
else
  if (selected_object_ptr->is_a() == OPERATOROBJECT) {
    if (handle_selected == true) {

      if (first_draw == true) {
selected_object_ptr->erase();
selected_object_ptr->unselect();
selected_object_ptr->draw(SOLID);
temp_operator_ptr->draw(DOTTED);
first_draw = false;
      } // first_draw
      else {
temp_operator_ptr->move_handle(x - x_state,
       y - y_state);
      }
    } // handle_selected
    else {
      if (first_draw == true)

//   Drawing the same thing twice in xor mode erases it. When
// moving an object, it is drawn once the first and last time,
// and twice afterwards

first_draw = false;
      else
temp_operator_ptr->draw(DOTTED);
      temp_operator_ptr->move(x - x_state, y - y_state);
      temp_operator_ptr->draw(DOTTED);
    }
  } // is_a OPERATAOROBJECT
  else {
    if (selected_object_ptr->is_a() == STREAMOBJECT) {
      if (handle_selected) {
if (first_draw == true) {
  conv_st_ptr =
    (StreamObject *) selected_object_ptr;
  conv_st_ptr->erase_handle();
  draw_handle(std_graphics_context, x, y);
  first_draw = false;
} // first_draw
else {
  draw_handle(std_graphics_context, x_state, y_state);
  draw_handle(std_graphics_context, x, y);
  selected_object_ptr->move_handle(x - x_state,
   y - y_state);
}
      } // handle_selected
    } // is_a STREAMOBJECT
  }

x_state = x;
y_state = y;
      } // selected_object_ptr != NULL
    } // tool_state == SELECT_TOOL

    // I-bar mode check
    temp_object_ptr = graphic_list.over(x, y);
    if (temp_object_ptr != NULL) {
      object_def = temp_object_ptr->is_a();
      if (object_def == OPERATOROBJECT) {

#ifdef GE_DEBUG
//          cerr << "It is an Operator Object" << endl;
#endif /* GE_DEBUG */

ibar_mode = true;
setcursor(drawing_a, True, XC_xterm);
      } // object_def == OPERATOROBJECT
      else
if (object_def == STREAMOBJECT) {

#ifdef GE_DEBUG
//            cerr << "It is an Stream Object" << endl;
#endif /* GE_DEBUG */

  ibar_mode = true;
  setcursor(drawing_a, True, XC_xterm);
} // object_def == STREAMOBJECT
else {
  ibar_mode = false;
  setcursor(drawing_a, False, None);
}
    } // temp_object_ptr != NULL
    else { // No object selected

#ifdef GE_DEBUG
```

```
//        cerr << "No object selected Object" << endl;
#endif /* GE_DEBUG */

      ibar_mode = false;
      setcursor(drawing_a, False, None);
    } // No object selected
  } else if (strcmp(args[0], "up") == 0) {
    if (tool_state == SELECT_TOOL)
      if (selected_object_ptr != NULL) {
if (text_selected) {
  if (first_draw == false) {
    draw_text_shadow(x_state, y_state,
     shadow_width, shadow_height);
    selected_object_ptr->text_locate(x, y);
            save_state(SAVE_REQUIRED);
  } // first_draw
} // text_selected
else
  if (selected_object_ptr->is_a() == OPERATOROBJECT) {
    if (first_draw == false) {
      temp_operator_ptr->draw(DOTTED);
      XYPAIR temp_pair = temp_operator_ptr->center();
      conv_op_ptr =
(OperatorObject *) selected_object_ptr;
      conv_op_ptr->radius(temp_operator_ptr->radius());
      conv_op_ptr->set_location(temp_pair.x,
temp_pair.y);
      if (handle_selected)
conv_op_ptr->set_default_text_location();
    } // first_draw
  } // is_a OPERATOROBJECT
  else
    if ((selected_object_ptr->is_a() == STREAMOBJECT)
       && (handle_selected)) {
      draw_handle(std_graphics_context, x_state, y_state);
    } // is_a STREAMOBJECT
if (drawing_changed == true) {
  graphic_list.move_notify(selected_object_ptr->is_a(),
   selected_object_ptr->id());
  graphic_list.draw();
  save_state(SAVE_REQUIRED);
  drawing_changed = false;
} // drawing_changed
handle_selected = false;
      } // selected_object_ptr != NULL
    first_draw = true;
  } else if (strcmp(args[0], "btn3down") == 0) {
    clear_status();
    if (ibar_mode == true) {
      if (object_def == OPERATOROBJECT) {
// op_being_updated = (OperatorObject *) temp_object_ptr;
operator_property_dialog(drawing_a, op_being_updated, x, y,
 graphic_list.cur_op_is_terminator(),
 graphic_list.avail_impl_langs_adr(),
 &graphic_list);
      }
      else
if (object_def == STREAMOBJECT) {
  stream_property_dialog(drawing_a, st_being_updated, x, y,
 &graphic_list);
  // XFlush(XtDisplay(drawing_a)); /* Stub for stream code */
}
else {
  XFlush(XtDisplay(drawing_a));/* Stub for Non Operator of Stream */
      }
    } // ibar_mode
  } else if (strcmp(args[0], "btn3motion") == 0) {
    temp_object_ptr = graphic_list.over(x, y);
    if (temp_object_ptr != NULL) {
      object_def = temp_object_ptr->is_a();
      if (object_def == OPERATOROBJECT) {

#ifdef GE_DEBUG
// cerr << "It is an Operator Object" << endl;
#endif /* GE_DEBUG */

ibar_mode = true;
setcursor(drawing_a, True, XC_xterm);

op_being_updated = (OperatorObject *) temp_object_ptr;
      }
      else
if (object_def == STREAMOBJECT) {

#ifdef GE_DEBUG
//   cerr << "It is an Stream Object" << endl;
#endif /* GE_DEBUG */

  ibar_mode = true;
  setcursor(drawing_a, True, XC_xterm);

  st_being_updated = (StreamObject *) temp_object_ptr;

}
else {
  ibar_mode = false;
  setcursor(drawing_a, True, XC_left_ptr);
}
    }
    else { // No object selected

#ifdef GE_DEBUG
      cerr << "No object selected Object" << endl;
#endif /* GE_DEBUG */

      ibar_mode = false;
```

```
      setcursor(drawing_a, False, None);
    }
  } else if (strcmp(args[0], "btn3up") == 0) {
    XFlush(XtDisplay(drawing_a)); /* Stub for stream code */
  } else if (strcmp(args[0], "motionnotify") == 0) {
    if (label_edit_mode == true) {
      label_edit_mode = false;
      if (ibar_object_ptr) {
if (ibar_object_ptr->is_a() == STREAMOBJECT) {
  labelName = ((StreamObject *)ibar_object_ptr)->name();
  warningMSG = (char *) malloc(strlen(labelName)+40);
  if (!valid_id(labelName)) {
    sprintf(warningMSG,"Invalid stream name: %s", labelName);
    warning(drawing_a, warningMSG);
    update_status(
       "Illegal stream name, retype: id ::= letter {alpha_numeric}",
       RING_BELL);
    ((StreamObject *)ibar_object_ptr)->erase_text();
    ((StreamObject *)ibar_object_ptr)->name("");
    ((StreamObject *)ibar_object_ptr)->draw_text(SOLID);
  } else if (is_keyword(labelName, false)) {
    sprintf(warningMSG,"Stream name is a keyword: %s", labelName);
    warning(drawing_a, warningMSG);
    update_status("Stream name is a keyword, retype", RING_BELL);
    ((StreamObject *)ibar_object_ptr)->erase_text();
    ((StreamObject *)ibar_object_ptr)->name("");
    ((StreamObject *)ibar_object_ptr)->draw_text(SOLID);
  } else {
    // Valid stream name, get any existing type information
    type_match = graphic_list.fetch_matching_stream_type(
 (StreamObject *)ibar_object_ptr, &state_change);
    if (state_change)
      ((StreamObject *)ibar_object_ptr)->draw(SOLID);
  }
  free(labelName);
  free(warningMSG);
}
else {
  labelName = ((OperatorObject *)ibar_object_ptr)->name();
  type_operator = (strchr(labelName,'.')) ? true : false;

  warningMSG = (char *) malloc(strlen(labelName)+80);
  if (!valid_op_id(labelName)) {
    sprintf(warningMSG,
      "Invalid operator name (syntax or keyword): %s", labelName);
    warning(drawing_a, warningMSG);
    update_status("Illegal operator name, retype:"
      " op_id ::= [id '.'] op_name ['(' [id_list] '|' [id_list] ')'",
      RING_BELL);
    ((OperatorObject *)ibar_object_ptr)->erase_text();
    ((OperatorObject *)ibar_object_ptr)->name("");
    ((OperatorObject *)ibar_object_ptr)->draw_text(SOLID);
  } else if (type_operator &&
      (((OperatorObject *)ibar_object_ptr)->is_composite())) {
    sprintf(warningMSG,
      "A Composite Operator can not be a Type: %s", labelName);
    warning(drawing_a, warningMSG);
    update_status("Composite Operator can not be a Type:"
  " rename operator or make Automic",
  RING_BELL);
    ((OperatorObject *)ibar_object_ptr)->erase_text();
    ((OperatorObject *)ibar_object_ptr)->name("");
    ((OperatorObject *)ibar_object_ptr)->draw_text(SOLID);
  } else if (!type_operator && !graphic_list.unique_op_id(labelName,
 ((OperatorObject *)ibar_object_ptr)->id())) {
    sprintf(warningMSG,
      "Simple Operator Names must be unique to level: %s", labelName);
    warning(drawing_a, warningMSG);
    update_status("Operators that are not types must have a "
      "unique name",
      RING_BELL);
    ((OperatorObject *)ibar_object_ptr)->erase_text();
    ((OperatorObject *)ibar_object_ptr)->name("");
    ((OperatorObject *)ibar_object_ptr)->draw_text(SOLID);
  }
  free(labelName);
}
      }
      string[0] = NULL;
      buffer[0] = NULL;
      ibar_object_ptr = NULL;
    }
    temp_object_ptr = graphic_list.over(x, y);
    if (temp_object_ptr != NULL) {
      object_def = temp_object_ptr->is_a();
      if (object_def == OPERATOROBJECT) {

#ifdef GE_DEBUG
//     cerr << "It is an Operator Object" << endl;
#endif /* GE_DEBUG */

ibar_mode = true;
setcursor(drawing_a, True, XC_xterm);

op_being_updated = (OperatorObject *) temp_object_ptr;
      }
      else
if (object_def == STREAMOBJECT) {

#ifdef GE_DEBUG
//       cerr << "It is an Stream Object" << endl;
#endif /* GE_DEBUG */

  ibar_mode = true;
  setcursor(drawing_a, True, XC_xterm);
```

```
  st_being_updated = (StreamObject *) temp_object_ptr;
}
else {
  ibar_mode = false;
  setcursor(drawing_a, True, XC_left_ptr);
}
    }
    else { // No object selected

#ifdef GE_DEBUG
//   cerr << "No object selected Object" << endl;
#endif /* GE_DEBUG */

      ibar_mode = false;
      setcursor(drawing_a, False, None);
    }
  } else if (strcmp(args[0], "key") == 0) {

#ifdef GE_DEBUG
//   cout << "key pressed: " << event->xkey.keycode << endl;
#endif

    count = XLookupString(&event->xkey, buffer,
  bufsize, &keysym, NULL);
    buffer[count] = NULL; /* add NULL terminator */

    if (label_edit_mode==true) {
      if ((keysym == XK_Return) || (keysym == XK_KP_Enter) ||
 (keysym == XK_Linefeed)) {
label_edit_mode = false;
if (ibar_object_ptr) {
  if (ibar_object_ptr->is_a() == STREAMOBJECT) {
    labelName = ((StreamObject *)ibar_object_ptr)->name();
    warningMSG = (char *) malloc(strlen(labelName)+40);
    if (!valid_id(labelName)) {
      sprintf(warningMSG,"Invalid stream name: %s", labelName);
      warning(drawing_a, warningMSG);
      update_status(
        "Illegal stream name, retype: id ::= letter {alpha_numeric}",
RING_BELL);
      ((StreamObject *)ibar_object_ptr)->erase_text();
      ((StreamObject *)ibar_object_ptr)->name("");
      ((StreamObject *)ibar_object_ptr)->draw_text(SOLID);
    } else if (is_keyword(labelName, false)) {
      sprintf(warningMSG,"Stream name is a keyword: %s", labelName);
      warning(drawing_a, warningMSG);
      update_status("Stream name is a keyword, retype", RING_BELL);
      ((StreamObject *)ibar_object_ptr)->erase_text();
      ((StreamObject *)ibar_object_ptr)->name("");
      ((StreamObject *)ibar_object_ptr)->draw_text(SOLID);
    } else {
      // Valid stream name, get any existing type information
      type_match = graphic_list.fetch_matching_stream_type(
   (StreamObject *)ibar_object_ptr, &state_change);
      if (state_change)
((StreamObject *)ibar_object_ptr)->draw(SOLID);
    }
    free(labelName);
    free(warningMSG);
  }
  else {
    labelName = ((OperatorObject *)ibar_object_ptr)->name();
    type_operator = (strchr(labelName,'.')) ? true : false;

    warningMSG = (char *) malloc(strlen(labelName)+80);
    if (!valid_op_id(labelName)) {
      sprintf(warningMSG,
 "Invalid operator name (syntax or keyword): %s", labelName);
      warning(drawing_a, warningMSG);
      update_status("Illegal operator name, retype: "
"op_id ::= [id '.'] op_name ['(' [id_list] '|' [id_list] ')'",
RING_BELL);
      ((OperatorObject *)ibar_object_ptr)->erase_text();
      ((OperatorObject *)ibar_object_ptr)->name("");
      ((OperatorObject *)ibar_object_ptr)->draw_text(SOLID);
    } else if ((strchr(labelName,'.') != NULL) &&
      (((OperatorObject *)ibar_object_ptr)->is_composite())) {
      sprintf(warningMSG,
      "A Composite Operator can not be a Type: %s", labelName);
      warning(drawing_a, warningMSG);
      update_status("Composite Operator can not be a Type:"
    " rename operator or make Automic",
    RING_BELL);
      ((OperatorObject *)ibar_object_ptr)->erase_text();
      ((OperatorObject *)ibar_object_ptr)->name("");
      ((OperatorObject *)ibar_object_ptr)->draw_text(SOLID);
    } else if (!type_operator && !graphic_list.unique_op_id(labelName,
 ((OperatorObject *)ibar_object_ptr)->id())) {
      sprintf(warningMSG,
      "Simple Operator Names must be unique to level: %s",
      labelName);
      warning(drawing_a, warningMSG);
      update_status("Operators that are not types must have a "
    "unique name",
    RING_BELL);
      ((OperatorObject *)ibar_object_ptr)->erase_text();
      ((OperatorObject *)ibar_object_ptr)->name("");
      ((OperatorObject *)ibar_object_ptr)->draw_text(SOLID);
    }
    free(labelName);
  }
}
string[0] = NULL;
buffer[0] = NULL;
ibar_object_ptr = NULL;
      }
```

```
      else
if (((keysym >= XK_KP_Space) && (keysym <= XK_KP_9)) ||
   ((keysym >= XK_space) && (keysym <= XK_asciitilde))) {
  if ((strlen(string) + strlen(buffer)) >= INPUT_LINE_SIZE) {
    XBell(display_ptr, 100);
  }
  else {
    strcat(string, buffer);
  }
}
else
  if ((keysym >= XK_Shift_L) && (keysym <= XK_Hyper_R)) {
    ; /* Do nothing because it's a modifier key */
  }
  else
    if ((keysym >= XK_F1) && (keysym <= XK_F35)) {
      if (buffer[0] != (char)NULL) {
if ((strlen(string) + strlen(buffer)) >= INPUT_LINE_SIZE) {
  XBell(display_ptr, 100);
}
else {
  strcat(string, buffer);
}
      }
    }
    else
      if ((keysym == XK_BackSpace) ||
 (keysym == XK_Delete)) {
if ((length = strlen(string)) > 0) {
  string[length - 1] = NULL;
}
else {
  XBell(display_ptr, 100);
}
      }

      temp_object_ptr = graphic_list.over(x, y);
      if (temp_object_ptr != NULL) {
object_def = temp_object_ptr->is_a();
if (label_edit_mode != false &&
   (object_def == OPERATOROBJECT ||
    object_def == STREAMOBJECT)) {

  ibar_object_ptr = temp_object_ptr;
  temp_object_ptr->erase_text();
  temp_object_ptr->name(string);
  temp_object_ptr->draw_text(SOLID);
  save_state(SAVE_REQUIRED);
//   temp_object_ptr->unselect();
}
      }
    }
    else
```

```
     if (alt_selected || ctrl_selected) {         // alt key pressed
alt_selected  = false;
ctrl_selected = false;
switch(keysym) {
case XK_D:                          // Decompose
case XK_d:
  handle_psdl_options(3);
  break;
case XK_P:                          // Goto Parent
case XK_p:
  handle_psdl_options(2);
  break;
case XK_R:                          // Goto Root
case XK_r:
  handle_psdl_options(1);
  break;
case XK_F:                          // Refresh Display
case XK_f:
  handle_edit_options(4);
  break;
case XK_Meta_L:                     // Alt key to activate
case XK_Meta_R:
  alt_selected = true;
  break;
case XK_Control_L:                  // Control key to activate
case XK_Control_R:
  ctrl_selected = true;
  break;
default:
  break;
}
      }
      else if (keysym == XK_Meta_L || keysym == XK_Meta_R) {
alt_selected = true;
      }
      else if (keysym == XK_Control_L || keysym == XK_Control_R) {
ctrl_selected = true;
      }
      else
if (selected_object_ptr != NULL) {
  if ((keysym == XK_BackSpace) ||
      (keysym == XK_Delete)) {
    selected_object_ptr->erase();
    OP_ID deleted_op_id = selected_object_ptr->id();
            save_state(SAVE_REQUIRED);
    graphic_list.delete_notify(selected_object_ptr->
       is_a(), deleted_op_id);
    selected_object_ptr->set_deleted();
    selected_object_ptr = NULL;
    graphic_list.draw();
    ibar_mode = false;
    setcursor(drawing_a, False, None);
  }
```

```
}
else
  if (ibar_mode==true &&
      label_edit_mode==false &&
      (((keysym >= XK_KP_Space) && (keysym <= XK_KP_9)) ||
       ((keysym >= XK_space) && (keysym <= XK_asciitilde)))) {
    if ((strlen(string) + strlen(buffer)) >= INPUT_LINE_SIZE) {
      XBell(display_ptr, 100);
    }
    else {
      strcat(string, buffer);
    }

    label_edit_mode = true;
    clear_status();

    temp_object_ptr = graphic_list.over(x, y);
    if (temp_object_ptr != NULL) {
      object_def = temp_object_ptr->is_a();
      if (object_def == OPERATOROBJECT ||
  object_def == STREAMOBJECT) {

//   temp_object_ptr->select();
ibar_object_ptr = temp_object_ptr;
temp_object_ptr->erase_text();
temp_object_ptr->name(string);
temp_object_ptr->draw_text(SOLID);
save_state(SAVE_REQUIRED);
//   temp_object_ptr->unselect();
      } // label_edit_mode != false && ()
    } // temp_object_ptr != NULL
  } // ibar_mode && label_edit_mode == false && ()
  } // strcmp KEY
} // draw


//   Callback function.  Just destroys the widget.

void widget_killer(Widget widget, XtPointer, XtPointer) {

  XtDestroyWidget(widget);
}

//   Callback function.  Called when Operator Tool button is
// pressed.

void op_button_cb(Widget, XtPointer, XtPointer) {

  select_state(OPERATOR_TOOL);

  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  if (selected_object_ptr != NULL) {
    selected_object_ptr->unselect();
    selected_object_ptr = NULL;
  }
}

//   Callback function.  Called when Terminator Tool button is
// pressed.

void term_button_cb(Widget, XtPointer, XtPointer) {

  select_state(TERMINATOR_TOOL);

  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  if (selected_object_ptr != NULL) {
    selected_object_ptr->unselect();
    selected_object_ptr = NULL;
  }
}


//   Null Callback.

void null_cb(Widget, XtPointer, XtPointer) {}


//  Callback function.  Called when Timer Tool OK button is
// pressed.  DL 8/22/96;  KBM 10/24/96

void timer_tool_ok_cb(Widget parent, XtPointer client_data,
      XtPointer call_data) {

  Widget              list_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs   = (XmAnyCallbackStruct *)call_data;
  int                 u_bound;
  XmString            *strlist;
  char                *text;
  ID_LIST             idp, timers;

  XtVaGetValues(list_w,
                XmNitemCount,   &u_bound,
                XmNitems,       &strlist,
                NULL);

  timers = NULL;
  if (u_bound > 0) {
    idp = (ID_LIST) malloc(sizeof(ID_NODE));
    idp->next = NULL;
    //if (XmStringGetLtoR(strlist[0], XmFONTLIST_DEFAULT_TAG, &text))    //@1
    if (XmStringGetLtoR(strlist[0], XmSTRING_DEFAULT_CHARSET, &text))  //@1
      idp->id = text;
    timers = idp;

    for (int i = 1; i < u_bound; i++) {
      idp->next = (ID_LIST) malloc(sizeof(ID_NODE));
```

```
      idp = idp->next;
      idp->next = NULL;
      //if (XmStringGetLtoR(strlist[i], XmFONTLIST_DEFAULT_TAG, &text))//@1
      if (XmStringGetLtoR(strlist[i], XmSTRING_DEFAULT_CHARSET, &text))//@1
idp->id = text;
    }
  }
  graphic_list.timer_list(timers);
  id_list_release(timers);       timers = NULL;

  XtDestroyWidget(XtParent(XtParent(XtParent(parent))));

}

//      Callback function.  Called when Timer Tool button is
// pressed.   DL 8/16/96.

void timers_button_cb(Widget parent,XtPointer client_data,XtPointer call_data){

  Widget   dialog, rc, pane, list, action_a;
  int      count = 0, i, n=0;
  ID_LIST  idp, timers;
  Arg      args[5];
  XmString *str,string;
  static ActionAreaItem action_items[] = {
    {"OK",     timer_tool_ok_cb,   NULL},
    {"Cancel", close_dialog,       NULL},
    {"Add",    timer_tool_add_cb,  NULL},
    {"Delete", timer_tool_del_cb,  NULL},
    {"Edit",   timer_tool_edit_cb, NULL},
    {"Help",   help_cb,            "timers_tool.hlp"}
  };

  //Build list for list widget
  timers = graphic_list.timer_list();

  idp = timers;
  while(idp) {
    count++;
    idp = idp->next;
  }
  idp = timers;
  str = (XmString *) XtMalloc (count * sizeof (XmString));
  for (i = 0; i < count; i++) {
    // str[i] = XmStringCreateLocalized(idp->id);     // @1
    str[i] = XmStringCreateSimple(idp->id);          // @1
    idp = idp->next;
  }
  id_list_release(timers);      timers = NULL;

  dialog = XtVaCreatePopupShell("dialog", xmDialogShellWidgetClass,
XtParent(parent), XmNtitle, "Timers Tool",
XmNdeleteResponse, XmDESTROY,
NULL);


  action_items[1].data = (XtPointer)dialog; //Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass, pane, NULL);
  // string = XmStringCreateLocalized("Enter or Edit Timers");   // @1
  string = XmStringCreateSimple("Enter or Edit Timers");         // @1
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  list = XmCreateScrolledList(rc, "Timer_List", NULL, 0);
  XtVaSetValues(list,
XmNvisibleItemCount, 10,
XmNitemCount,        count,
XmNitems,            str,
NULL);
  XtManageChild(list);
  for(i = 0; i < count; i++)
    XmStringFree(str[i]);

  XtManageChild(rc);

  //Set client data for "OK", "Add", "Del", and "Edit" buttons
  action_items[0].data = (XtPointer)list;
  action_items[2].data = (XtPointer)list;
  action_items[3].data = (XtPointer)list;
  action_items[4].data = (XtPointer)list;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  XtManageChild(pane);
  XtPopup(dialog, XtGrabNone);

}


//      Callback function.  Called when Informal Description Tool OK is
// pressed.  Added by Doug Lange 8/19/96.

static void inform_tool_ok_pushed(Widget w, XtPointer client_data,
  XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;
```

```
  char *text = XmTextGetString(text_w);
  graphic_list.graph_informal_desc(text);
  free(text);

  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}

//      Callback function.  Called when Informal Description Tool Button is
// pressed.  Added by Doug Lange 8/19/96.

static void informal_button_cb(Widget w, XtPointer client_data,
     XtPointer call_data)              {

  Widget    dialog, pane, rc, text_w, action_a;
  XmString  string;
  char      *description;

  static ActionAreaItem   action_items[] = {
    {"OK",     inform_tool_ok_pushed, NULL                    },
    {"Cancel", close_dialog,          NULL                    },
    {"Help",   help_cb,               "inform_tool.hlp"  }
  };

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 XtParent(w),
 XmNtitle, "Informal Design Description",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog;   //Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass, pane, NULL);
  string = XmStringCreateSimple("Enter or Edit Informal Description");     //01
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  description = graphic_list.graph_informal_desc();

  int     n = 0;
  Arg     args[10];
  XtSetArg(args[n], XmNrows,                 12); n++;
  XtSetArg(args[n], XmNcolumns,              70); n++;
  XtSetArg(args[n], XmNscrollVertical,       true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,     false); n++;
  XtSetArg(args[n], XmNeditMode,             XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,             true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,             true); n++;
  XtSetArg(args[n], XmNvalue,                description); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);
  //text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
  //      rc, NULL);

  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);

}

//   Callback function.  Called when Stream Tool button is
// pressed.

void stream_button_cb(Widget, XtPointer, XtPointer) {

  select_state(STREAM_TOOL);

  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  if (selected_object_ptr != NULL) {
    selected_object_ptr->unselect();
    selected_object_ptr = NULL;
  }
}

//   Callback function.  Called when Select Tool button is
// pressed.

void select_button_cb(Widget, XtPointer, XtPointer) {

  select_state(SELECT_TOOL);

  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}


static void types_tool_ok_pushed(Widget w, XtPointer client_data,
```

```
 XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);
  char *org_text = graphic_list.global_types();

  if (strcmp(text, org_text) != 0) {
    graphic_list.global_types(text);
    save_state(SAVE_REQUIRED);
  }

  free(text);              text     = NULL;
  free(org_text);          org_text = NULL;

  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}


void types_button_cb(Widget w, XtPointer client_data,
     XtPointer call_data)             {

  Widget    dialog, pane, rc, text_w, action_a;
  XmString  string;
  char      *description;

  static ActionAreaItem   action_items[] = {
    {"OK",     types_tool_ok_pushed,  NULL                    },
    {"Cancel", close_dialog,          NULL                    },
    {"Help",   help_cb,               "types_tool.hlp"  }
  };

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 XtParent(w),
 XmNtitle, "Prototype Types Specification",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog;   //Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass, pane, NULL);
  string = XmStringCreateSimple("View or Edit Prototype Types Specification");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  description = graphic_list.global_types();

  int     n = 0;
  Arg     args[10];
  XtSetArg(args[n], XmNrows,                  12); n++;
  XtSetArg(args[n], XmNcolumns,               70); n++;
  XtSetArg(args[n], XmNscrollVertical,        true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,      true); n++;
  XtSetArg(args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,              true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,              true); n++;
  XtSetArg(args[n], XmNvalue,                 description); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);
  //text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
  //      rc, NULL);

  //  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);
  //  Note:  If you have problems with '}' symbols in the text, uncomment
  //  the line above.

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);

}

static void spec_tool_ok_pushed(Widget w, XtPointer client_data,
  XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);
  char *org_text = graphic_list.cur_op_spec();

  if (strcmp(text, org_text) != 0) {
    graphic_list.cur_op_spec(text);
    save_state(SAVE_REQUIRED);
  }
```

```
  free(text);              text     = NULL;
  free(org_text);          org_text = NULL;

  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}

void spec_button_cb(Widget w, XtPointer client_data,
    XtPointer call_data)              {

  Widget    dialog, pane, rc, text_w, action_a;
  XmString  string;
  char      *description;

  static ActionAreaItem   action_items[] = {
    {"OK",     spec_tool_ok_pushed, NULL                    },
    {"Cancel", close_dialog,          NULL                    },
    {"Help",   help_cb,               "spec_tool.hlp"  }
  };

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 XtParent(w),
 XmNtitle, "Prototype Specification",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog;   //Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass, pane, NULL);
  string = XmStringCreateSimple("View or Edit Prototype Specification");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  description = graphic_list.cur_op_spec();

  int     n = 0;
  Arg     args[10];
  XtSetArg(args[n], XmNrows,                12); n++;
  XtSetArg(args[n], XmNcolumns,             70); n++;
  XtSetArg(args[n], XmNscrollVertical,      true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,    true); n++;
  XtSetArg(args[n], XmNeditMode,            XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,            true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,            true); n++;
  XtSetArg(args[n], XmNvalue,               description); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);
  //text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
  //      rc, NULL);

  //  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);
  //  Note:  If you have problems with '}' symbols in the text, uncomment
  //  the line above.

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);


}


//   Callback function.  Called when the radio buttons in the
// properties dialog box are pushed.  Called twice: once to
// unselect old button, again to select the new one.

void radio_box_cb(Widget, XtPointer which,
                  XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if (state->set) {
    if ((int) which == 0)
      state_stream = false;
    else
      state_stream = true;
  }
}


void save_indicator_cb(Widget widget, XtPointer,
       XtPointer cb_struct_ptr) {

  if (psdl_modified)
    handle_file_options(0);   // save

}
```

```
void error_indicator_cb(Widget widget, XtPointer client_data,
       XtPointer call_data) {

  if ((errors_present == NULL) || (!syntax_checked))
    handle_psdl_options(0);  // check syntax
  else {
    report_errors(errors_present, toplevel, next_action_ptr,
  &return_sde_flag, &prev_status);
  }

}


//   If graph_editor is invoked in viewer mode, this function
// handles ClientMessage events from the syntax-directed editor.
// Commented-out code handles data passed in a property, which
// this version of the editor doesn't take advantage of.
// Used during testing, and left in for future use, if necessary.

void event_handler(Widget widget, XtPointer,
                   XEvent* in_event, Boolean*) {

  char buffer[INPUT_LINE_SIZE];
  Display *display_ptr = XtDisplayOfObject(widget);
  Window window = DefaultRootWindow(display_ptr);
//  char **data;
//  int return_count;
//  XTextProperty text_prop_return;
//  Atom property_name;
  char message_in[30];


  strcpy(message_in, in_event->xclient.data.b);
  if (strcmp(message_in, "GEDATAIN") == 0) {

    graphic_list.build_from_sde(gdnode);        // @3

#ifdef GE_DEBUG
    //    printf("graphic_list: after build\n");
    //    graphic_list.summarize();
#endif

    //  graphic_list.build_from_disk();    // @3
    graphic_list.draw();
  }
  else if (strcmp(message_in, "PrintWindow") == 0)
    {
      if (PrintCmd.op == Snd_to_Prt) {
if ((PrintCmd.printer != NULL) && (*PrintCmd.printer != '\0')) {
  sprintf(buffer,
  "xwd -frame -id %d | xpr -gray 2 -device ps | lpr -P%s ",
  XtWindow(toplevel), PrintCmd.printer);
}
else {
  sprintf(buffer,
  "xwd -frame -id %d | xpr -gray 2 -device ps | lpr ",
  XtWindow(toplevel));
}
setcursor(toplevel,True,XC_watch);
system(buffer);
setcursor(toplevel,True,XC_left_ptr);
      }
      else {
if ((PrintCmd.file != NULL) && (*PrintCmd.file != '\0')) {
  sprintf(buffer,
  "xwd -frame -id %d > %s ",
  XtWindow(toplevel), PrintCmd.file);
}
else {
  warning(drawing_a,"A file name must be suppled.");
}
setcursor(toplevel,True,XC_watch);
system(buffer);
setcursor(toplevel,True,XC_left_ptr);
      }
    }
  else {
#ifdef GE_DEBUG
//      cout << "Event " << message_in << endl;
#endif
    }
}


void set_current_op() {

  char *cur_op_name;

  cur_op_name = graphic_list.current_op_name();
  if (cur_op_name != NULL)
    XtVaSetValues(current_op_name, XmNvalue, cur_op_name, NULL);
  free(cur_op_name);
}

void set_current_op_met() {

  char buffer[25] = "MET ";
  char *time;
  char *met = buffer;

  if (graphic_list.cur_op_spec_met() != UNDEFINED_TIME) {
    time = time_with_units(graphic_list.cur_op_spec_met(),
   graphic_list.cur_op_spec_met_unit());
    strncat(met,time,20);
    XtVaSetValues(current_op_met, XmNvalue, met, NULL);
    free(time);
```

```
  }
  else
    XtVaSetValues(current_op_met, XmNvalue, "", NULL);
}

void set_editor_title() {

  char   title_str[63] = "PSDL Editor: \0";
  char   *title_ptr = title_str;
  char   *name_ptr;

  name_ptr = graphic_list.root_op_name();
  if (name_ptr != NULL)
    strncat(title_ptr,name_ptr,50);
  XtVaSetValues(toplevel, XmNtitle, title_ptr, NULL);
  free(name_ptr);
}


void init_motif() {

  // Simulated arguments
  //  char*  args[] = {"edit_graph","-geometry","800x600",NULL};
  //  int    signed_argc = 3;
  //  char** argv = args;
  char   title_str[63] = "PSDL Editor: \0";
  char   *title_ptr = title_str;
  XmString tmp;

  print_event = (XEvent *) malloc(sizeof(XEvent));     // @7

  toplevel = XtVaAppInitialize(&app, "edit_graph", options,
                               XtNumber(options), &Global_argc, Global_argv,
                               NULL, NULL);

  display_ptr = XtDisplay(toplevel);
  XtGetApplicationResources(toplevel, (XtPointer) &Resrcs,
                            resources, XtNumber(resources),
                            NULL, 0);
  screen_ptr = XtScreen(toplevel);
  initialize_color_table(screen_ptr);
  root_window = RootWindowOfScreen(screen_ptr);
  gcv1.foreground = BlackPixelOfScreen(screen_ptr);
  gcv1.background = WhitePixelOfScreen(screen_ptr);
  gcv2.foreground = BlackPixelOfScreen(screen_ptr);
  gcv2.background = WhitePixelOfScreen(screen_ptr);
  gcv3.foreground = WhitePixelOfScreen(screen_ptr);
  gcv3.background = WhitePixelOfScreen(screen_ptr);
  gc_mask = GCForeground | GCBackground;
  std_graphics_context = XCreateGC(display_ptr,
   root_window, gc_mask, &gcv1);
  dotted_context = XCreateGC(display_ptr,
     root_window, gc_mask, &gcv2);
```

```
erase_context = XCreateGC(display_ptr, root_window, gc_mask,
                          &gcv3);
XSetLineAttributes(display_ptr, dotted_context, 1,
                   LineOnOffDash, CapButt, JoinMiter);
XSetFunction(display_ptr, dotted_context, GXxor);

main_w = XtVaCreateManagedWidget("main_w", xmFormWidgetClass,
 toplevel, NULL);
build_menu_bar(main_w, menubar);
XtManageChild(menubar);
rowcol =
  XtVaCreateManagedWidget("rowcol", xmRowColumnWidgetClass,
  main_w,
  XmNnumColumns, 1,
  //  XmNorientation,  XmHORIZONTAL,
  NULL);

make_buttons(rowcol,
     op_button, term_button, stream_button, select_button,
     types_button, spec_button,
     timers_button, informal_button,
     op_button_pixmap, term_button_pixmap,
     stream_button_pixmap, select_button_pixmap,
     types_button_pixmap, spec_button_pixmap,
     informal_button_pixmap, timers_button_pixmap,
     display_ptr, screen_ptr);

XtAddCallback(op_button,       XmNactivateCallback, op_button_cb,       NULL);
XtAddCallback(term_button,     XmNactivateCallback, term_button_cb,     NULL);
XtAddCallback(stream_button,   XmNactivateCallback, stream_button_cb,   NULL);
XtAddCallback(select_button,   XmNactivateCallback, select_button_cb,   NULL);
XtAddCallback(types_button,    XmNactivateCallback, types_button_cb,    NULL);
XtAddCallback(spec_button,     XmNactivateCallback, spec_button_cb,     NULL);
XtAddCallback(timers_button,   XmNactivateCallback, timers_button_cb,   NULL);
XtAddCallback(informal_button, XmNactivateCallback, informal_button_cb,NULL);

XtVaSetValues(toplevel, XmNtitle, title_ptr, NULL);

current_op_name =
  XtVaCreateManagedWidget("current_op_name", xmTextWidgetClass,
  main_w,
  XmNvalue, "",
  XmNshadowThickness, 1,
  NULL);

current_op_met =
  XtVaCreateManagedWidget("current_op_met", xmTextWidgetClass,
  main_w,
  XmNvalue, "",
  XmNwidth, 150,
  XmNshadowThickness, 1,
  NULL);
```

```
  scrolled_win =
    XtVaCreateManagedWidget("scrolled_win",
    xmScrolledWindowWidgetClass,
    main_w,
    // XmNwidth, 1200,
    // XmNheight, 750,
    XmNscrollingPolicy, XmAUTOMATIC,
    XmNscrollBarDisplayPolicy, XmAS_NEEDED,
    NULL);
  actions.string = "draw";
  actions.proc = draw;
  XtAppAddActions(app, &actions, 1);
  status_indicator =
    XtVaCreateManagedWidget("satus_indicator", xmTextWidgetClass,
    main_w,
    XmNheight, 31,
    XmNvalue, "",
    NULL);

  save_indicator =
    XtVaCreateManagedWidget("save_indicator",
    xmDrawnButtonWidgetClass,
    main_w,
    XmNrecomputeSize, false,
    XmNpushButtonEnabled, false,
    XmNshadowType, XmSHADOW_IN,
    XmNwidth, 120,
    XmNheight, 31,
    XmNmarginBottom, 13,
    XmNlabelType, XmSTRING,
    NULL);
  XtAddCallback(save_indicator, XmNactivateCallback, save_indicator_cb,NULL);

  error_indicator =
    XtVaCreateManagedWidget("error_indicator",
    xmDrawnButtonWidgetClass,
    main_w,
    XmNrecomputeSize, false,
    XmNpushButtonEnabled, false,
    XmNshadowType, XmSHADOW_IN,
    XmNwidth, 120,
    XmNheight, 31,
    XmNmarginBottom, 13,
    XmNlabelType, XmSTRING,
    NULL);
  XtAddCallback(error_indicator, XmNactivateCallback,
error_indicator_cb, NULL);

  drawing_a =
    XtVaCreateManagedWidget("drawing_a",
    xmDrawingAreaWidgetClass, scrolled_win,
    XmNunitType, Xm1000TH_INCHES,
    XmNwidth, 11000,
    XmNheight, 8500,
    XmNresizePolicy, XmNONE,
    NULL);
  XtAddCallback(drawing_a, XmNexposeCallback, redraw, NULL);

  XtVaSetValues(drawing_a, XmNunitType, XmPIXELS, NULL);
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  XtVaGetValues(drawing_a, XmNwidth, &width, XmNheight, &height,
                NULL);
  drawing_area_pixmap = XCreatePixmap(display_ptr,
      root_window, width, height,
      DefaultDepthOfScreen(screen_ptr));
  XFillRectangle(display_ptr, drawing_area_pixmap,
                 erase_context, 0, 0, width, height);

  XtVaSetValues(drawing_a,  XmNtranslations,
XtParseTranslationTable(translations), NULL);

  XtVaSetValues(rowcol,
XmNtopAttachment,    XmATTACH_FORM,
XmNrightAttachment,  XmATTACH_NONE,
XmNleftAttachment,   XmATTACH_FORM,
XmNbottomAttachment, XmATTACH_WIDGET,
XmNbottomWidget,     save_indicator,
NULL);

  XtVaSetValues(menubar,
XmNtopAttachment,    XmATTACH_FORM,
XmNrightAttachment,  XmATTACH_FORM,
XmNleftAttachment,   XmATTACH_WIDGET,
XmNleftWidget,       rowcol,
XmNbottomAttachment, XmATTACH_NONE,
NULL);

  XtVaSetValues(current_op_name,
XmNtopAttachment,    XmATTACH_WIDGET,
XmNtopWidget,        menubar,
XmNrightAttachment,  XmATTACH_WIDGET,
XmNrightWidget,      current_op_met,
XmNleftAttachment,   XmATTACH_WIDGET,
XmNleftWidget,       rowcol,
XmNbottomAttachment, XmATTACH_NONE,
NULL);

  XtVaSetValues(current_op_met,
XmNtopAttachment,    XmATTACH_WIDGET,
XmNtopWidget,        menubar,
XmNrightAttachment,  XmATTACH_FORM,
XmNleftAttachment,   XmATTACH_NONE,
XmNbottomAttachment, XmATTACH_NONE,
NULL);

  XtVaSetValues(scrolled_win,
```

```
XmNtopAttachment,    XmATTACH_WIDGET,
XmNtopWidget,        current_op_name,
XmNrightAttachment,  XmATTACH_FORM,
XmNleftAttachment,   XmATTACH_WIDGET,
XmNleftWidget,       rowcol,
XmNbottomAttachment, XmATTACH_WIDGET,
XmNbottomWidget,     status_indicator,
NULL);

  XtVaSetValues(save_indicator,
XmNtopAttachment,    XmATTACH_NONE,
XmNrightAttachment,  XmATTACH_NONE,
XmNleftAttachment,   XmATTACH_FORM,
XmNbottomAttachment, XmATTACH_FORM,
NULL);

  XtVaSetValues(error_indicator,
XmNtopAttachment,    XmATTACH_NONE,
XmNrightAttachment,  XmATTACH_NONE,
XmNleftAttachment,   XmATTACH_WIDGET,
XmNleftWidget,       save_indicator,
XmNbottomAttachment, XmATTACH_FORM,
NULL);

  XtVaSetValues(status_indicator,
XmNtopAttachment,    XmATTACH_NONE,
XmNrightAttachment,  XmATTACH_FORM,
XmNleftAttachment,   XmATTACH_WIDGET,
XmNleftWidget,       error_indicator,
XmNbottomAttachment, XmATTACH_FORM,
NULL);


  XtRealizeWidget(toplevel);
  draw_window = XtWindow(drawing_a);

  graphic_list.set_draw_environ(display_ptr,
std_graphics_context,
erase_context, dotted_context,
draw_window,
&drawing_area_pixmap,
color_table,
width, height);
  graphic_list.set_error_tgt(drawing_a);

  set_current_op();
  set_current_op_met();

  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);

  toplevel_window = XtWindowOfObject(toplevel);
  Atom display_id_atom = XInternAtom(display_ptr, "WINDOW_ID",
     False);
  XChangeProperty(display_ptr, root_window, display_id_atom,
  XA_WINDOW, 32, PropModeReplace,
  (unsigned char *) &toplevel_window, 1);
  XtAddEventHandler(toplevel, NoEventMask, true, event_handler,
    NULL);

  motif_initialized = true;
}

//   translations provides the mappings for the keyboard
// mapping table that allow the drawing canvas to capture
// mouse and keyboard events.

// The primary function, edit_graph.  Modified from original main() by
// Doug Lange 9/9/96


/*********************************************************************
* this method is added to support the edit graph and sde change over.
* now the edit graph module is not a standalone but a method called
* from the sde
*********************************************************************/

//extern "C" {

int edit_graph(GRAPH_DESC current_graph, ACTION next_action,
       ERROR_MSGS sde_error_msgs)                    {    //@2

  XEvent event;  // added for custom main loop

  int    reply;
  Quest_Script delete_script  =
         {"","Deleted operators will be purged?", "Ok","No","Cancel", BTN1};

  next_action_ptr   = next_action;
  errors_present    = sde_error_msgs;
  return_sde_flag = false;

  gdnode = current_graph;

  // motif_initialized assumed to be false at start of procedure

  if (motif_initialized) {
    XFillRectangle(display_ptr, drawing_area_pixmap,
   erase_context, 0, 0, width, height);
    XFillRectangle(display_ptr, draw_window,
   erase_context, 0, 0, width, height);
    if (gdnode) {
      graphic_list.build_from_sde(gdnode);
    }

    if (save_performed)
      save_state(NOT_MODIFIED);
```

```
  if (prev_status) {
    update_status(prev_status, false);
    free(prev_status);
    prev_status = NULL;
  }
  graphic_list.draw();
  setcursor(toplevel,True,XC_left_ptr);
}
else {
  init_motif();
  prev_status = NULL;
  save_state(NOT_MODIFIED);
  save_performed = false;
  default_color = WHITE;
  default_font  = COURIERBOLD12;
  graphic_list.set_default_font(default_font);
  if (gdnode) {
    graphic_list.build_from_sde(gdnode);
  }

  graphic_list.draw();

  select_state(SELECT_TOOL);

  // Initialize printer command
  PrintCmd.op      = Snd_to_Prt;
  PrintCmd.printer = dup_str("");
  PrintCmd.file    = dup_str("");
  PrintCmd.answer  = 0;

  // and event
  print_event->type = ClientMessage;
  print_event->xclient.window = toplevel_window;
  print_event->xclient.format = 8;
  strcpy(print_event->xclient.data.b, "PrintWindow");
}
set_editor_title();
set_current_op();
set_current_op_met();

syntax_checked = true;    // syntax is checked on each entry to editor
error_label();

if (graphic_list.cur_op_is_terminator())
  XtVaSetValues(op_button, XmNsensitive, False, NULL);
else
  XtVaSetValues(op_button, XmNsensitive, True,  NULL);

printf("\n"); //flushes the event queue
XFlush(display_ptr);
```

```
#ifdef GE_DEBUG
      cout << "Starting Motif event loop" << endl;
#endif

  selected_object_ptr = NULL;

  // Custom main loop to check for return to sde
  do {
    XtAppNextEvent(app, &event);
    XtDispatchEvent(&event);

    if (return_sde_flag) {
      if (graphic_list.has_deleted()) {
  reply = AskUser(app, drawing_a, delete_script);
  if (reply != YES)
    return_sde_flag = false;
      }
    }

  } while (return_sde_flag == false);

  if ((next_action->option != REVERT) &&
      (next_action->option != ABANDON))
    graphic_list.write_to_sde(gdnode);

  if ((next_action_ptr->option == SAVE_TO_DISK) ||
      (next_action_ptr->option == REVERT)) // not really saved, but not
    save_performed = true;                 // modified
  else
    save_performed = false;                // assume need to save for abandon

  // If we are not coming back, kill the window
  if (!next_action_ptr->reinvoke) {
    XtUnrealizeWidget(toplevel);
    XFlush(display_ptr);
  }
  else {  // otherwise, will be returning, erase window and exit
    setcursor(toplevel,True,XC_watch);
  }
  return 0;
}

//} // extern "C"


/************************************************************************
*                    --- end of graph_editor.C
************************************************************************/
```

```
/* ************************************************************

Name:          graph_object.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the GraphObject class.

           The GraphObject is the base class for the main graph
         objects displayed in the graph editor. It is not
         designed to be directly instantiated, so most of its
         methods are virtual.

           There are a number of static class variables used
         to store graphic information necessary for the
         descendents to draw themselves.

History:

  @1   96/10/06  Ken Moeller
       Now pass units explicitly, not encoded in time.

  @2   96/10/11  Ken Moelelr
       Size of color_table was stepping on _drawing_area_pixmap.
       Had to enlarge the table to MAXCOLORS+1
************************************************************ */
#ifndef graph_object_h
#define graph_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include "ge_defs.h"
#include "font_table.h"

class GraphObjectList;

class GraphObject {
protected:
  static Display *_display_ptr;
  static  GC _graphics_context, _erase_context, _dotted_context;
  static Window _draw_window;     .
  static Pixmap *_drawing_area_pixmap;
  static unsigned long _color_table[MAXCOLORS+1]; ´     // @2
  static FontTable *_font_table;
  static int _default_font;
  static Widget _error_tgt;

  GraphObject *_next_ptr;

public:

  static Dimension window_width, window_height;
```



```
  static void set_draw_environ(Display *display_ptr,
                               GC graphics_context,
                               GC erase_context,
                               GC dotted_context,
                               Window draw_window,
                               Pixmap *drawing_area_pixmap,
                               unsigned long color_table[],
                               Dimension width,
                               Dimension height);
  static void font_init(Display *display_ptr);
  static void set_default_font(int font_id)
                                    {_default_font = font_id;}
  static int font_text_width(int font_id, char *in_string);
  static int font_text_height(int font_id);
  static char *font_name(int font_id)
                  {return _font_table->font_name(font_id);}
  static void set_font(GC graphics_context, int font_id);
  static void set_error_tgt(Widget widget) {_error_tgt = widget;}

  GraphObject();
  virtual ~GraphObject() {
#ifdef GE_DEBUG
//    printf("GraphObject base class Destructor: delete _next_ptr\n");
#endif
    delete _next_ptr;}
  void link(GraphObject *next_object) {_next_ptr = next_object;}
  GraphObject* next() {return _next_ptr;}
  virtual GE_STATUS build_from_property();
  virtual GE_STATUS build_from_disk(FILE *) {return FAILED;}
  virtual GE_STATUS write_to_property() {return FAILED;}
  virtual GE_STATUS write_to_disk(FILE *) {return FAILED;}
  virtual void draw(DRAW_STYLE) {}
  virtual void select() {}
  virtual void unselect() {}
  virtual void erase() {}
  virtual BOOLEAN hit(int, int) {return false;}
  virtual BOOLEAN over(int, int) {return false;} // Added 8/25/96, dha
  virtual CLASS_DEF is_a() {return GRAPHOBJECT;}
  virtual void set_object_ptrs(GraphObjectList *) {}
  virtual OP_ID id() {return UNDEFINED_OPNUM;}
  virtual char *name() {return "";}
  virtual void set_location(int, int) {}
  virtual void set_deleted() {}
  virtual void delete_notify(CLASS_DEF, OP_ID) {}
  virtual void name(char *) {}
  virtual void set_modified() {}
  virtual BOOLEAN text_selected() {return false;}
  Display *display_ptr() {return _display_ptr;}
  GC graphics_context() {return _graphics_context;}
  Window draw_window() {return _draw_window;}
  Pixmap *drawing_area_pixmap() {return _drawing_area_pixmap;}
  virtual void move_notify(CLASS_DEF, OP_ID) {}
  virtual void move_handle(int, int) {}
```

```
virtual BOOLEAN hit_handle(int, int) {return false;}
virtual void color(COLOR) {}
virtual void set_object_font(int) {};
virtual void erase_text() {}
virtual void move_text(int, int) {}
virtual void draw_text(DRAW_STYLE) {}
virtual BOOLEAN is_deleted() {return false;}
virtual void undelete_notify(CLASS_DEF, OP_ID) {}
virtual void reset_handles_drawn_state() {}
  virtual int text_width() {return 0;}
  virtual int text_height() {return 0;}
  virtual void text_locate(int, int) {}
};

#endif;
```

```
/* ************************************************************

Name:          graph_object.C
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 11 Sep 92
Remarks:   Implementation for the GraphObject class.

          The GraphObject is the base class for the main graph
         objects displayed in the graph editor. It is not
         designed to be directly instantiated, so most of its
         methods are virtual.

Credits:    Portions of code are adapted from the following:

          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.

History:

  @1   96/10/06  Ken Moeller

  @2   96/10/11  Ken Moeller
       Size of color_table had to be enlarged.

************************************************************ */
#include <Xm/MessageB.h>
#include <stream.h>
#include "graph_object.h"

//   Initializers for the static class variables.

int GraphObject::_default_font = 0;
Widget GraphObject::_error_tgt = NULL;

//  K. Moeller  8/2/96
//  added definitions for the class static variables
Window GraphObject::_draw_window;
Display *GraphObject::_display_ptr;
GC GraphObject::_dotted_context;
GC GraphObject::_erase_context;
GC GraphObject::_graphics_context;
Pixmap *GraphObject::_drawing_area_pixmap;
FontTable *GraphObject::_font_table;
unsigned long GraphObject::_color_table[MAXCOLORS+1];   // @2
Dimension GraphObject::window_width;
Dimension GraphObject::window_height;


GraphObject::GraphObject() {
  _next_ptr = NULL;
}

//   Sets up the drawing environment for the graph objects.

void GraphObject::set_draw_environ(Display *display_ptr,
                                   GC graphics_context,
                                   GC erase_context,
                                   GC dotted_context,
                                   Window draw_window,
                                   Pixmap *drawing_area_pixmap,
                                   unsigned long color_table[],
                                   Dimension width,
                                   Dimension height) {
  int i;

  GraphObject::_display_ptr = display_ptr;
  GraphObject::_graphics_context = graphics_context;
  GraphObject::_erase_context = erase_context;
  GraphObject::_dotted_context = dotted_context;
  GraphObject::_draw_window = draw_window;
  GraphObject::_drawing_area_pixmap = drawing_area_pixmap;
  for(i = 0; i <= MAXCOLORS; i++)
    GraphObject::_color_table[i] = color_table[i];
  GraphObject::window_width = width;
  GraphObject::window_height = height;
}

//   Initializes the font table.

void GraphObject::font_init(Display *display_ptr) {

  _font_table = new FontTable();
  _font_table->init(display_ptr);
}

//  Returns the width in pixels of the requested font.

int GraphObject::font_text_width(int font_id, char *in_string) {

  return _font_table->width(font_id, in_string);
}

//   Returns the height in pixels of the requested font.

int GraphObject::font_text_height(int font_id) {
```

```
  return _font_table->height(font_id);
}

//   Sets the requested font as the drawing font.

void GraphObject::set_font(GC graphics_context, int font_id) {

  XSetFont(_display_ptr, graphics_context,
           _font_table->font_id(font_id));
}
```

```
GE_STATUS GraphObject::build_from_property() {return FAILED;}
```

```
/* ************************************************************

Name:          graph_object_list.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the GraphObjectList class.

           A GraphObjectList is a linked list of graphic
         objects, implemented as GraphObjects. Currently,
         a GraphObject may be either an OperatorObject or
         a StreamObject.

           Although not currently implemented this way, most
         of the functionality necessary to draw the graph title
         on the drawing canvas is present.

History:

  @1   96/10/04  Ken Moeller
       Removed un-needed routines to read and write to disk.

  @2   96/10/04  Ken Moeller
       Removed un-needed routines to read and write to property.

************************************************************* */
#ifndef graph_object_list_h
#define graph_object_list_h 1

#include <X11/Intrinsic.h>
#include "ge_defs.h"
#include "ge_interface.h"
#include "ge_utilities.h"
#include "graph_object.h"
#include "stream_object.h"

class GraphObject;

class GraphObjectList {
  protected:
    Display   *_display_ptr;
    GC        _graphics_context, _erase_context, _dotted_context;
    Pixmap    *_drawing_area_pixmap;
    Window    _draw_window;
    Dimension _width, _height;
    Widget    _error_tgt;
    BOOLEAN   _psdl_modified;

//----------  ge_interface.h data  ---------------

    char* _root_op_name;
    int   _root_op_num;
    char* _parent_op_name;
    int   _parent_op_num;
    char* _current_op_name;
    int   _current_op_num;

    GraphObject *_head_input_list;
    GraphObject *_head_output_list;

    char* _cur_op_spec;

    int     _cur_op_spec_met;
    int     _cur_op_spec_met_unit;
    int     _cur_op_is_terminator;

    GraphObject *_head_ptr;

    ID_LIST _timer_list;

    char*   _graph_informal_desc;

    ID_LIST _avail_impl_langs;

    char*   _global_types;

//---------------------------------------------

    void set_text_dimensions();
    void draw_handles(GC draw_context, int x1, int y1, int x2, int y2);

  public:
    GraphObjectList();
    virtual ~GraphObjectList();
    GE_STATUS build_from_sde(GRAPH_DESC gd);
    GE_STATUS write_to_sde(GRAPH_DESC gd);

    void draw();
    void erase();

    GraphObject* hit(int x, int y);
    GraphObject* over(int x, int y); // Added 8/26/96, dha

    CLASS_DEF is_a() {return GRAPHOBJECTLIST;}
    GraphObject* target_object(CLASS_DEF object_type, OP_ID id);
    OP_ID request_id(CLASS_DEF class_type);
    void add(GraphObject *new_object_ptr);

    void set_draw_environ(Display *display_ptr,
                          GC graphics_context, GC erase_context,
                          GC dotted_context, Window draw_window,
                          Pixmap *drawing_area_pixmap,
                          unsigned long color_table[],
                          Dimension width, Dimension height);
    void set_error_tgt(Widget widget);
```

```
  void get_del_op_list(char *del_op_str[], OP_ID del_op_id[],
                       int &num_del_ops);
  void set_undeleted(CLASS_DEF class_type, OP_ID id);
  void delete_notify(CLASS_DEF class_type, OP_ID deleted_obj_id);

  void move_notify(CLASS_DEF object_type, OP_ID object_id);

  void set_default_font(int font_id);
  char *font_name(int font_id)
                   {return GraphObject::font_name(font_id);}

  BOOLEAN has_deleted();

  void set_psdl_modified()   { _psdl_modified = true; }
  void reset_psdl_modified() { _psdl_modified = false; }
  BOOLEAN psdl_modified()    { return _psdl_modified; }

  BOOLEAN unique_op_id(char *searchName, int VertexNum);
  void propagate_stream(OP_ID st_id, BOOLEAN redraw);
  BOOLEAN fetch_matching_stream_type(StreamObject *toPtr,
     BOOLEAN *state_change);

//--------------  Methods to get values of ge_interface.h  ---------
//
// Note that copies of strings are returned, not the original string.
// Make sure that you free this memory when no longer required.
//

  char *root_op_name()        { return dup_str(_root_op_name); }
  int   root_op_num()         { return _root_op_num; }
  char *parent_op_name()      { return dup_str(_parent_op_name); }
  int   parent_op_num()       { return _parent_op_num; }
  char *current_op_name()     { return dup_str(_current_op_name); }
  int   current_op_num()      { return _current_op_num; }

  GraphObject *input_list()   { return _head_input_list; }
  GraphObject *output_list()  { return _head_output_list; }
  GraphObject *cur_graph()    { return _head_ptr; }

  char *cur_op_spec()         { return dup_str(_cur_op_spec); }
  int   cur_op_spec_met()     { return _cur_op_spec_met; }
  int   cur_op_spec_met_unit(){ return _cur_op_spec_met_unit; }
  int  cur_op_is_terminator() { return _cur_op_is_terminator; }

    ID_LIST timer_list()        { return id_list_copy(_timer_list); }

    char *global_types()        { return dup_str(_global_types); }

    char *graph_informal_desc() { return dup_str(_graph_informal_desc); }

    ID_LIST avail_impl_langs_adr()  { return _avail_impl_langs; }

  //--------------  Methods to set values of ge_interface.h  ---------
  //
  // Note that copies of strings are stored, not the original string.
  // Make sure that you free this memory when no longer required.
  //

    void root_op_name(char *ptr);
    void root_op_num(int op_num)         { _root_op_num    = op_num; }
    void parent_op_name(char *ptr);
    void parent_op_num(int op_num)       { _parent_op_num  = op_num; }
    void current_op_name(char *ptr);
    void current_op_num(int op_num)      { _current_op_num = op_num; }

    void cur_op_spec(char *spec_ptr);
    void global_types(char *type_ptr);
    void cur_op_spec_met(int met)        { _cur_op_spec_met      = met; }
    void cur_op_spec_met_unit(int met_u) { _cur_op_spec_met_unit = met_u; }
    void cur_op_is_terminator(int term)  { _cur_op_is_terminator = term; }

    void timer_list(ID_LIST x)           {id_list_replace(&_timer_list, x); }

    void graph_informal_desc(char *ptr);

  //-----------------------------------------------------------------

    void release();

    void summarize();
    void summarize_List_GraphObject(GraphObject *ptr);
};

#endif
```

```
/* ************************************************************

Name:          graph_object_list.C
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 11 Sep 92
Remarks:   Implementation of a GraphObjectList.

          A GraphObjectList is a linked list of graphic
         objects, implemented as GraphObjects. Currently,
         a GraphObject may be either an OperatorObject or
         a StreamObject.

Credits:    Portions of code are adapted from the following:

          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.

Changes:
        9/94    add synchronization code to the build_from_disk() routine
                so that it will creates the gedatatransfile.lock file before
                reading from the gedatatransfile.txt file and remove the
                gedatatransfile.lock file when the reading is completed.

                codes are also added to the store_data() routine in the
                ge_interface.c file so that it will only write to
                gedatatransfile.txt file if the gedatatransfile.lock
                does not exist in current directory.

Reengineering:
Doug Lange added build_from_sde member function on 9/10/96

History:

  @1   96/09/29  Ken Moeller
       Read in streams as well as operators from sde.

  @2   96/10/04  Ken Moeller
       Removed un-needed routines to read and write to disk.

  @3   96/10/06  Ken Moeller
       Removed un-needed routines to read and write to property.

  @4   96/10/06  Ken Moeller
       Added streams to objects written out in write_to_sde.

************************************************************ */
#include <stdio.h>
#include <string.h>
#include <stream.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <Xm/MessageB.h>
#include "graph_object_list.h"
#include "operator_object.h"
#include "stream_object.h"
#include "warning.h"
#include "ge_utilities.h"
#include "ge_utilities_debug.h"
#include "get_unique_id.h"


BOOLEAN fetch_stream_type(GraphObject  *GraphList,
  StreamObject *toPtr,
  BOOLEAN      *state_change);

void inherit_stream_type(GraphObject *GraphList,
 StreamObject *fromPtr,
 BOOLEAN redraw);


GraphObjectList::GraphObjectList() {

  _root_op_name              = NULL;
  _parent_op_name            = NULL;
  _current_op_name           = NULL;
  _head_input_list           = NULL;
  _head_output_list          = NULL;
  _cur_op_spec               = NULL;
  _head_ptr                  = NULL;
  _timer_list                = NULL;
  _graph_informal_desc       = NULL;
  _avail_impl_langs          = NULL;
  _global_types              = NULL;

  _display_ptr               = NULL;
  _graphics_context          = NULL;
  _erase_context             = NULL;
  _dotted_context            = NULL;
  _drawing_area_pixmap       = NULL;
  _draw_window               = 0;
  _error_tgt                 = NULL;

  _width  = 0;
  _height = 0;
  _psdl_modified = false;
```

```
}


GraphObjectList::~GraphObjectList() {

#ifdef GE_DEBUG
    printf("GraphObjectList Destructor\n");
#endif

    release();

}


GE_STATUS GraphObjectList::build_from_sde(GRAPH_DESC gd) {
  char       *error_str_ptr;
  GraphObject *temp_object_ptr, *current_ptr;
  GE_STATUS    status = SUCCEEDED;
  OP_LIST      op_ptr;
  ST_LIST      st_ptr;
  ID_LIST      id_ptr;

  error_str_ptr = NULL;

  release();

  _root_op_name   = dup_str(gd->root_op_name);
  _root_op_num    = gd->root_op_num;

  _parent_op_name = dup_str(gd->parent_op_name);
  _parent_op_num  = gd->parent_op_num;

  _current_op_name = dup_str(gd->current_op_name);
  _current_op_num  = gd->current_op_num;

  st_ptr = gd->input_list;
  while (st_ptr != NULL) {
    temp_object_ptr = new StreamObject(st_ptr->st);
    st_ptr = st_ptr->next;

    if(_head_input_list == NULL) {
      _head_input_list = temp_object_ptr;
      current_ptr      = _head_input_list;
    }
    else {
      current_ptr->link(temp_object_ptr);
      current_ptr = current_ptr->next();
    }
  }

  st_ptr = gd->output_list;
  while (st_ptr != NULL) {
    temp_object_ptr = new StreamObject(st_ptr->st);
    st_ptr = st_ptr->next;

    if(_head_output_list == NULL) {
      _head_output_list = temp_object_ptr;
      current_ptr       = _head_output_list;
    }
    else {
      current_ptr->link(temp_object_ptr);
      current_ptr = current_ptr->next();
    }
  }

_cur_op_spec               = dup_str(gd->cur_op_spec);
_cur_op_spec_met           = gd->cur_op_spec_met;
_cur_op_spec_met_unit      = gd->cur_op_spec_met_unit;
_cur_op_is_terminator      = gd->cur_op_is_terminator;

op_ptr = gd->operator_list;
while(op_ptr != NULL) {
  temp_object_ptr = new OperatorObject(op_ptr->op);
  if(_head_ptr == NULL) {
    _head_ptr   = temp_object_ptr;
    current_ptr = _head_ptr;
  }
  else {
    current_ptr->link(temp_object_ptr);
    current_ptr = current_ptr->next();
  }
  op_ptr = op_ptr->next;
}

//   Operators are read in; time for the streams

st_ptr = gd->stream_list;
while (st_ptr != NULL) {
  temp_object_ptr = new StreamObject(st_ptr->st);
  st_ptr = st_ptr->next;
  temp_object_ptr->set_object_ptrs(this);
  if(_head_ptr == NULL) {
    _head_ptr   = temp_object_ptr;
    current_ptr = _head_ptr;
  }
  else {
    current_ptr->link(temp_object_ptr);
    current_ptr = current_ptr->next();
  }
}

_timer_list = id_list_copy(gd->timer_list);
_avail_impl_langs = id_list_copy(gd->avail_impl_langs);

_graph_informal_desc = dup_str(gd->graph_informal_desc);
_global_types        = dup_str(gd->global_types);
```

```
  _psdl_modified = false;

  return(SUCCEEDED);            // @1
}


//Written by Doug Lange; modified by Ken Moeller
GE_STATUS GraphObjectList::write_to_sde(GRAPH_DESC gd) {

  GraphObject  *go;
  StreamObject *stPtr;
  OPERATOR      op      = NULL;
  OP_LIST       op_list = NULL;
  STREAM        st      = NULL;
  ST_LIST       st_list = NULL;

  go    = NULL;
  stPtr = NULL;


  graph_release(gd);

  gd->root_op_name   = dup_str(_root_op_name);
  gd->root_op_num    = _root_op_num;

  gd->parent_op_name = dup_str(_parent_op_name);
  gd->parent_op_num  = _parent_op_num;

  gd->current_op_name = dup_str(_current_op_name);
  gd->current_op_num  = _current_op_num;

  go      = _head_input_list;
  st_list = NULL;
  while (go != NULL) {
    stPtr = (StreamObject *) go;
    if (st_list == NULL ) {
      st_list = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
      gd->input_list = st_list;
    }
    else {
      st_list->next = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
      st_list = st_list->next;
    }
    st_list->st = stPtr->clone(NULL);
    st_list->next = NULL;
    go = go->next();
  }

  go      = _head_output_list;
  st_list = NULL;
  while (go != NULL) {
    stPtr = (StreamObject *) go;
    if (st_list == NULL ) {
      st_list = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
      gd->output_list = st_list;
    }
    else {
      st_list->next = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
      st_list = st_list->next;
    }
    st_list->st = stPtr->clone(NULL);
    st_list->next = NULL;
    go = go->next();
  }

  gd->cur_op_spec_met        = _cur_op_spec_met;
  gd->cur_op_spec_met_unit   = _cur_op_spec_met_unit;

  gd->cur_op_spec            = dup_str(_cur_op_spec);

  gd->cur_op_is_terminator   = _cur_op_is_terminator;

  // Build the operator and stream lists in two passes, first to get
  // the operators, then the streams.  This is so that the operators
  // will exist in memory when the stream from and to pointers are assigned
  go = _head_ptr;
  op_list = NULL;
  while (go != NULL) {
    if (go->is_a() == OPERATOROBJECT) {
      if (op_list == NULL ) {
op_list = (OP_LIST) malloc(sizeof(OP_LIST_NODE));   // kbm
gd->operator_list = op_list;
      }
      else {
op_list->next = (OP_LIST) malloc(sizeof(OP_LIST_NODE));   // kbm
op_list = op_list->next;
      }
      op_list->op = ((OperatorObject *) go)->clone();
      op_list->next = NULL;
    }
    go = go->next();
  }

  st_list = NULL;
  go = _head_ptr;
  op_list = NULL;
  while (go != NULL) {
    if (go->is_a() == STREAMOBJECT) {          // @4
      if (st_list == NULL ) {
st_list = (ST_LIST) malloc(sizeof(ST_LIST_NODE));    // kbm
gd->stream_list = st_list;
      }
      else {
st_list->next = (ST_LIST) malloc(sizeof(ST_LIST_NODE));    // kbm
st_list = st_list->next;
```

```
      }
      st_list->st = ((StreamObject *) go)->clone(gd->operator_list);
      st_list->next = NULL;
    }
    go = go->next();
  }

  gd->timer_list          = id_list_copy(_timer_list);
  gd->avail_impl_langs    = id_list_copy(_avail_impl_langs);

  gd->graph_informal_desc = dup_str(_graph_informal_desc);
  gd->global_types        = dup_str(_global_types);

  return SUCCEEDED;
}


//   Draws the GraphObjectList to the drawing canvas.

void GraphObjectList::draw() {
  GraphObject *temp_object_ptr;

  temp_object_ptr = _head_ptr;

  XFillRectangle(_display_ptr, _draw_window, _erase_context, 0,
                 0, _width, _height);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 _erase_context, 0, 0, _width, _height);

//   Streams are drawn first to prevent the operators' handles
// from overwriting the arrowheads.  Handles are drawn in xor
// mode so they will erase properly.

//   Since handles are drawn in xor mode, it's important to
// track whether they've been drawn or not.  Redrawing the
// handles unintentionally erases them.

  while(temp_object_ptr != NULL) {
    if(temp_object_ptr->is_a() == STREAMOBJECT) {
      temp_object_ptr->reset_handles_drawn_state();
      temp_object_ptr->draw(SOLID);
    }
    temp_object_ptr = temp_object_ptr->next();
  }
  temp_object_ptr = _head_ptr;
  while(temp_object_ptr != NULL) {
    if(temp_object_ptr->is_a() == OPERATOROBJECT) {
      temp_object_ptr->reset_handles_drawn_state();
      temp_object_ptr->draw(SOLID);
    }
    temp_object_ptr = temp_object_ptr->next();
  }
}
```

```
void GraphObjectList::erase() {}

//   If the given coordinates are located inside one of the
// graph objects or their text strings, the function returns
// a pointer to the object.

GraphObject* GraphObjectList::hit(int x, int y) {
  GraphObject *temp_object_ptr;

  temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    if(temp_object_ptr->hit(x, y))
      return temp_object_ptr;
    else
      temp_object_ptr = temp_object_ptr->next();
  }
  return (GraphObject *) NULL;
}

GraphObject* GraphObjectList::over(int x, int y) {
  GraphObject *temp_object_ptr;

  temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    if(temp_object_ptr->over(x, y))
      return temp_object_ptr;
    else
      temp_object_ptr = temp_object_ptr->next();
  }
  return (GraphObject *) NULL;
}

//   Returns a pointer to the requested object.

GraphObject* GraphObjectList::target_object(
                                CLASS_DEF object_type, OP_ID id) {
  GraphObject* temp_object_ptr;

  temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    if((temp_object_ptr->is_a() == object_type) &&
       (temp_object_ptr->id() == id)) {
      return temp_object_ptr;
    }
    temp_object_ptr = temp_object_ptr->next();
  }
  warning(_error_tgt,"Requested operator id not found");
  return NULL;
}
```

```
//   Notifies all the objects that one of them has been
// deleted, so that they may take appropriate actions.

void GraphObjectList::delete_notify(CLASS_DEF class_type,
                                    OP_ID deleted_obj_id) {
  GraphObject *temp_object_ptr;

  temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    temp_object_ptr->delete_notify(class_type, deleted_obj_id);
    temp_object_ptr = temp_object_ptr->next();
  }
}

/*****************************************************************
 * Obtain the next available unique id from the server.
 *****************************************************************/
OP_ID GraphObjectList::request_id(CLASS_DEF class_type) {

  OP_ID id;

  id = get_unique_id();

  return id;
}

//   Adds a new GraphObject to the list.

void GraphObjectList::add(GraphObject *new_object_ptr) {
  GraphObject *temp_object_ptr;

  temp_object_ptr = _head_ptr;

  if(_head_ptr == NULL)
    _head_ptr = new_object_ptr;
  else
    if(_head_ptr->next() == NULL)
      _head_ptr->link(new_object_ptr);
    else {
      while(temp_object_ptr->next() != NULL)
        temp_object_ptr = temp_object_ptr->next();
      temp_object_ptr->link(new_object_ptr);
    }
}

//   Sets the necessary drawing variables, and performs the
// same operation for the GraphObject class.

void GraphObjectList::set_draw_environ(Display *display_ptr,
                GC graphics_context, GC erase_context,
                GC dotted_context, Window draw_window,
                Pixmap *drawing_area_pixmap,
                unsigned long color_table[], Dimension width,
                Dimension height) {

  _display_ptr         = display_ptr;
  _graphics_context    = graphics_context;
  _erase_context       = erase_context;
  _dotted_context      = dotted_context;
  _draw_window         = draw_window;
  _drawing_area_pixmap = drawing_area_pixmap;
  _width               = width;
  _height              = height;
  GraphObject::set_draw_environ(_display_ptr, _graphics_context,
              _erase_context, _dotted_context, _draw_window,
              _drawing_area_pixmap, color_table, width, height);
  GraphObject::font_init(_display_ptr);
}

//   Notifies the objects that one of them has been moved.

void GraphObjectList::move_notify(CLASS_DEF object_type,
                                  OP_ID object_id) {
  GraphObject *temp_object_ptr;

  temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    temp_object_ptr->move_notify(object_type, object_id);
    temp_object_ptr = temp_object_ptr->next();
  }
}

//   Sets the default font.

void GraphObjectList::set_default_font(int font_id) {

  GraphObject::set_default_font(font_id);
}

//   Draws a handle on the graph.

void GraphObjectList::draw_handles(GC draw_context, int x1,
                                   int y1, int x2, int y2) {

  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y2 - HANDLESIZE, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
```

```
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y1,
                 HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y2 - HANDLESIZE,
                 HANDLESIZE, HANDLESIZE);
}


//   Makes a list of deleted operators.

void GraphObjectList::get_del_op_list(char *del_op_str[],
                                      OP_ID del_op_id[],
                                      int &num_del_ops) {
  GraphObject *temp_obj_ptr;
  int index = 0;

  temp_obj_ptr = _head_ptr;

  while((temp_obj_ptr != NULL) && (index < MAXDELETEDOPS)) {
    if((temp_obj_ptr->is_a() == OPERATOROBJECT) &&
       (temp_obj_ptr->is_deleted())) {
         del_op_str[index] = dup_str(temp_obj_ptr->name());
         del_op_id[index] = temp_obj_ptr->id();
         index++;
     }
     temp_obj_ptr = temp_obj_ptr->next();
  }
  num_del_ops = index;
}

//   Notifies the objects that the given object has been
// undeleted.

void GraphObjectList::set_undeleted(CLASS_DEF class_type,
                                    OP_ID id) {
  GraphObject *temp_obj_ptr;

  temp_obj_ptr = _head_ptr;

  while(temp_obj_ptr != NULL) {
    temp_obj_ptr->undelete_notify(class_type, id);
    temp_obj_ptr = temp_obj_ptr->next();
  }
}

//   Sets the widget used to display the error message box.
```



```
void GraphObjectList::set_error_tgt(Widget widget) {

  _error_tgt = widget;
  GraphObject::set_error_tgt(widget);
  SplineObject::set_error_tgt(widget);
  FontTable::set_error_tgt(widget);
}


void GraphObjectList::root_op_name(char *ptr) {

  free(_root_op_name);
  _root_op_name = dup_str(ptr);
}


void GraphObjectList::parent_op_name(char *ptr) {

  free(_parent_op_name);
  _parent_op_name = dup_str(ptr);
}


void GraphObjectList::current_op_name(char *ptr) {

  free(_current_op_name);
  _current_op_name = dup_str(ptr);
}


void GraphObjectList::cur_op_spec(char *spec_ptr) {

  free(_cur_op_spec);
  _cur_op_spec = dup_str(spec_ptr);
}


void GraphObjectList::global_types(char *type_ptr) {

  free(_global_types);
  _global_types = dup_str(type_ptr);
}


void GraphObjectList::graph_informal_desc(char *ptr) {

  free(_graph_informal_desc);
  _graph_informal_desc = dup_str(ptr);
}


void GraphObjectList::release() {
```

```
#ifdef GE_DEBUG
  printf("GraphObjectList::release\n");
#endif

  free(_root_op_name);
  _root_op_name = NULL;

  free(_parent_op_name);
  _parent_op_name = NULL;

  free(_current_op_name);
  _current_op_name = NULL;

  delete _head_input_list;
  _head_input_list = NULL;
  delete _head_output_list;
  _head_output_list = NULL;

  free(_cur_op_spec);
  _cur_op_spec = NULL;

  delete _head_ptr;
  _head_ptr = NULL;

  id_list_release(_timer_list);
  _timer_list = NULL;

  free(_graph_informal_desc);
  _graph_informal_desc = NULL;

  id_list_release(_avail_impl_langs);
  _avail_impl_langs = NULL;

  free(_global_types);
  _global_types = NULL;

}

void GraphObjectList::summarize_List_GraphObject(GraphObject *ptr) {

  GraphObject *temp;
  StreamObject *tempST;
  OperatorObject *tempOP;

  char *name;

  temp = ptr;
  while (temp != NULL) {
    if (temp->is_a() == OPERATOROBJECT) {
      tempOP = (OperatorObject *) temp;
      name = tempOP->name();
      printf("    Operator ID %d %s\n", tempOP->id(), name);
      free(name);
    }
    else {
      tempST = (StreamObject *) temp;
      name = tempST->name();
      printf("    Stream ID %d %s\n", tempST->id(), name);
      free(name);
    }
    temp = temp->next();
  }
}

void GraphObjectList::summarize() {

  printf("GraphObjectList Summary:\n");

  printf("  root_op_name:    \"%s\"\n",_root_op_name);
  printf("  parent_op_name:  \"%s\"\n",_parent_op_name);
  printf("  current_op_name: \"%s\"\n",_current_op_name);
  printf("  graph_informal_desc: \"%s\"\n", _graph_informal_desc);

  printf("\n  INPUT LIST:   %s\n",
 (_head_input_list) ? "" : "NULL");
  summarize_List_GraphObject(_head_input_list);

  printf("\n  OUTPUT LIST:   %s\n",
 (_head_output_list) ? "" : "NULL");
  summarize_List_GraphObject(_head_output_list);

  printf("\n  OPERATOR/STREAM LIST: %s\n",
 (_head_ptr) ? "" : "NULL");
  summarize_List_GraphObject(_head_ptr);

  printf("\n  TIMER LIST:    %s\n",
 (_timer_list) ? "" : "NULL");
  if (_timer_list) summarize_ID_LIST(_timer_list);

  printf("\n  IMPL_LANG_LIST\t%s\n",
  (_avail_impl_langs) ? "" : "NULL");
  if (_avail_impl_langs) summarize_ID_LIST(_avail_impl_langs);

  printf("End of GraphObjectList\n");

}

BOOLEAN GraphObjectList::has_deleted() {

  GraphObject *tmpPtr;

  tmpPtr = _head_ptr;

  while (tmpPtr) {
    if ((tmpPtr->is_a() == OPERATOROBJECT) &&
(((OperatorObject *) tmpPtr)->is_deleted()))
```

```
      return true;

    tmpPtr = tmpPtr->next();
  }

  return false;
}


BOOLEAN GraphObjectList::fetch_matching_stream_type(StreamObject *toPtr,
    BOOLEAN *state_change) {

  *state_change = false;

  return fetch_stream_type(_head_ptr,         toPtr, state_change) ||
         fetch_stream_type(_head_input_list,  toPtr, state_change) ||
         fetch_stream_type(_head_output_list, toPtr, state_change);

}


BOOLEAN fetch_stream_type(GraphObject  *GraphList,
  StreamObject *toPtr,
  BOOLEAN      *state_change) {

  GraphObject  *tmpPtr;
  StreamObject *stPtr;
  char         *searchName, *stName;
  OP_ID         searchID;

  // We need the name to compare with
  searchName = toPtr->name();
  searchID   = toPtr->id();

  // Now search the GraphList for the first matching stream
  tmpPtr = GraphList;
  while (tmpPtr) {
    if ((tmpPtr->is_a() == STREAMOBJECT) &&
(((StreamObject *) tmpPtr)->id() != searchID)) {
      stPtr = (StreamObject *) tmpPtr;

      // grab the name and check for match
      stName = stPtr->name();

      if (strcmp(stName, searchName) == 0) {
if (stPtr->is_state_variable() != toPtr->is_state_variable())
  *state_change = true;

toPtr->inherit_type(stPtr);
free(stName);
free(searchName);
return true;
      }
      free(stName);
    }
    tmpPtr = tmpPtr->next();
  }
  free(searchName);
  return false;
}

BOOLEAN GraphObjectList::unique_op_id(char *searchName, int VertexNum) {

  GraphObject  *tmpPtr;
  char         *opName;

  // Search through all of the current graph
  tmpPtr = _head_ptr;
  while (tmpPtr) {
    if ((tmpPtr->is_a() == OPERATOROBJECT) &&
(((OperatorObject *) tmpPtr)->id() != VertexNum)) {
      opName = ((OperatorObject *) tmpPtr)->name();
      if (strcmp(opName, searchName) == 0) {
free(opName);
return false;
      }
      free(opName);
    }
    tmpPtr = tmpPtr->next();
  }
  return true;
}

void GraphObjectList::propagate_stream(OP_ID st_id, BOOLEAN redraw) {

  GraphObject  *tmpPtr;
  StreamObject *fromPtr;

  // Create a pointer to stream from which we will propagate
  tmpPtr = _head_ptr;;
  while (tmpPtr) {
    if ((tmpPtr->is_a() == STREAMOBJECT) &&
(((StreamObject *) tmpPtr)->id() == st_id)) {
      fromPtr = (StreamObject *) tmpPtr;
      break;  // jump out of loop
    }
    tmpPtr = tmpPtr->next();
  }

  inherit_stream_type(_head_ptr,         fromPtr, redraw);
  inherit_stream_type(_head_input_list,  fromPtr, false);
  inherit_stream_type(_head_output_list, fromPtr, false);

}
```

```
void inherit_stream_type(GraphObject *GraphList,
 StreamObject *fromPtr,
 BOOLEAN redraw) {

  GraphObject  *tmpPtr;
  StreamObject *stPtr;
  BOOLEAN       old_state_value;
  char         *fromName, *stName;

  // We need the name to compare with
  fromName = fromPtr->name();

  // Now update all the streams that match by name
  tmpPtr = GraphList;
  while (tmpPtr) {
    if ((tmpPtr->is_a() == STREAMOBJECT) &&
(((StreamObject *) tmpPtr)->id() != fromPtr->id())) {
      stPtr = (StreamObject *) tmpPtr;

      // grab the name and check for match
      stName = stPtr->name();

      if (strcmp(stName, fromName) == 0) {
old_state_value = stPtr->is_state_variable();
if (redraw && (fromPtr->is_state_variable() != old_state_value))
  stPtr->erase();
stPtr->inherit_type(fromPtr);
if (redraw && (fromPtr->is_state_variable() != old_state_value))
  stPtr->draw(SOLID);
      }
      free(stName);
    }

    tmpPtr = tmpPtr->next();
  }
  free(fromName);
}
```

```
#ifndef inter_process_utilities_h
#define inter_process_utilities_h 1

#include "ge_interface.h"


#define CHKWORD      0xFACE


#ifdef __cplusplus
extern "C" {
#endif

typedef struct xfer_buffer {
  char *Buf;
  int  Idx;
  int  Max;
} XferBuffer;

/* static variables for inter-process communication */
int
        sde_to_ge_channel[2],
        ge_to_sde_channel[2];

XferBuffer
*xfer;

#ifdef _NO_PROTO

void    create_xfer_buf();
BOOLEAN enlarge_xfer_buf();
BOOLEAN collapse_xfer_buf();
BOOLEAN xfer_space_avail();

int     read_xfer();
int     write_xfer();

BOOLEAN packString();
BOOLEAN packInteger();
BOOLEAN packBoolean();
BOOLEAN packID_LIST();

char*   unpackString();
int     unpackInteger();
BOOLEAN unpackBoolean();
ID_LIST unpackID_LIST();

BOOLEAN packGraphNames();
void    unpackGraphNames();

BOOLEAN readChkWord();
BOOLEAN writeChkWord();

BOOLEAN synch_read();
BOOLEAN synch_write();

BOOLEAN readAction();
BOOLEAN writeAction();

BOOLEAN readGraphDesc();
BOOLEAN writeGraphDesc();

BOOLEAN readErrorMsgs();
BOOLEAN writeErrorMsgs();

#else

void    create_xfer_buf(XferBuffer **xfer);
BOOLEAN enlarge_xfer_buf(XferBuffer *xfer);
BOOLEAN collapse_xfer_buf(XferBuffer *xfer);
BOOLEAN xfer_space_avail(XferBuffer *xfer, int delta);

int read_xfer(int Chl, XferBuffer *xfer);
int write_xfer(int Chl, XferBuffer *xfer);

BOOLEAN packString(char* str, XferBuffer *xfer);
BOOLEAN packInteger(int inInt, XferBuffer *xfer);
BOOLEAN packBoolean(BOOLEAN Bool, XferBuffer *xfer);
BOOLEAN packID_LIST(ID_LIST idList, XferBuffer *xfer);

char*   unpackString(XferBuffer *xfer);
int     unpackInteger(XferBuffer *xfer);
BOOLEAN unpackBoolean(XferBuffer *xfer);
ID_LIST unpackID_LIST(XferBuffer *xfer);

BOOLEAN readChkWord(int *chkWord, int Chl);
BOOLEAN writeChkWord(int chkWord, int Chl);

BOOLEAN synch_read(int Chl);
BOOLEAN synch_write(int Chl);

BOOLEAN packGraphNames(GRAPH_DESC graph, XferBuffer *xfer);
void    unpackGraphNames(GRAPH_DESC graph, XferBuffer *xfer);

BOOLEAN readAction(ACTION next_action, XferBuffer *xfer, int Chl);
BOOLEAN writeAction(ACTION next_action, XferBuffer *xfer, int Chl);

BOOLEAN readGraphDesc(GRAPH_DESC graph, XferBuffer *xfer, int Chl);
BOOLEAN writeGraphDesc(GRAPH_DESC graph, XferBuffer *xfer, int Chl);

BOOLEAN readErrorMsgs(ERROR_MSGS *errs, XferBuffer *xfer, int Chl);
BOOLEAN writeErrorMsgs(ERROR_MSGS errs, XferBuffer *xfer, int Chl);

#endif

#ifdef __cplusplus
```

```
}
#endif

#endif
```

```
#include <string.h>
#include <memory.h>
#include "inter_process_utilities.h"
#include "ge_utilities.h"


#ifdef __cplusplus
extern "C" {
#endif

#define INIT_XFER_BUF 256
#define MAX_PACKET    4096

#ifdef _NO_PROTO
void create_xfer_buf(xfer)
XferBuffer **xfer;
{
#else
void create_xfer_buf(XferBuffer **xfer) {
#endif

  XferBuffer *Temp;

  /* Allocate space for the XferBuffer */
  Temp = (XferBuffer *) malloc(sizeof(XferBuffer));

  /* Now allocate the buffer and initialize */
  Temp->Buf = (char *) malloc(INIT_XFER_BUF);
  Temp->Max = INIT_XFER_BUF;
  Temp->Idx = 0;
  *Temp->Buf = '\0';

  *xfer = Temp;

}

#ifdef _NO_PROTO
BOOLEAN enlarge_xfer_buf(xfer)
XferBuffer *xfer;
{
#else
BOOLEAN enlarge_xfer_buf(XferBuffer *xfer) {
#endif

  char *Temp;

  if (Temp = (char *) malloc(2*(xfer->Max))) {
    memcpy(Temp, xfer->Buf, xfer->Max);
    free(xfer->Buf);
    xfer->Buf = Temp;
    xfer->Max = 2*(xfer->Max);

    return true;
  }
  else {
#ifdef GE_DEBUG
    printf("Error in enlarge_xfer_buf\n");
#endif
    return false;
  }
}


#ifdef _NO_PROTO
BOOLEAN collapse_xfer_buf(xfer)
XferBuffer *xfer;
{
#else
BOOLEAN collapse_xfer_buf(XferBuffer *xfer) {
#endif

  free(xfer->Buf);
  xfer->Idx = 0;
  xfer->Max = INIT_XFER_BUF;
  xfer->Buf = (char *) malloc(xfer->Max);

  if (xfer->Buf != NULL)
    return true;
  else
    return false;

#ifdef GE_DEBUG
    printf("Error in enlarge_xfer_buf\n");
#endif
    return false;
}


#ifdef _NO_PROTO
BOOLEAN xfer_space_avail(xfer, delta)
XferBuffer *xfer;
int        delta;
{
#else
BOOLEAN xfer_space_avail(XferBuf *xfer, int delta) {
#endif

  BOOLEAN too_small;

  while ((too_small = ((xfer->Idx + delta) > (xfer->Max))) &&
 enlarge_xfer_buf(xfer))
    ;

#ifdef GE_DEBUG
  if (too_small)
    printf("Error in xfer_space_avail\n");
```

```
#endif

  return !too_small;
}


#ifdef _NO_PROTO
int read_xfer(Chl, xfer)
int        Chl;
XferBuffer *xfer;
{
#else
int read_xfer(int Chl, XferBuffer *xfer) {
#endif

  BOOLEAN  xfer_status;
  int      int_size, xfer_size, actual_size, tot_size;
  int      num_packets;
  int      packetIdx;
  char    *packet_ptr;


  int_size = sizeof(int);
  tot_size = 0;

  /* First determine how many packets will be required */
  actual_size = read(Chl, &num_packets, int_size);
  if (actual_size != int_size) {
    printf("ERROR:  read_xfer not able to transfer buffer (num_packets).\n");
    return 0;
  }

#ifdef GE_DEBUG
  printf("read_xfer %d packets...", num_packets);
#endif

  xfer->Idx = 0;   /* Clear to get accurate xfer_space_avail */

  /* Next get data */
  for (packetIdx = 1; packetIdx <= num_packets; packetIdx++) {

    /* Data is sent in a size, data pair */
    actual_size = read(Chl, &xfer_size, int_size);
    if (actual_size != int_size) {
      printf("ERROR:  read_xfer not able to receive buffer (packet size).\n");
      return 0;
    }

    /* Make sure we have room to place the packet */
    xfer_status = xfer_space_avail(xfer, xfer_size);
    if (!xfer_status) {
      printf("ERROR:  read_xfer not able to allocate sufficient memory.\n");
      return 0;
    }

    /* We may have changed the location of buffer, get new buffer_ptr */
    packet_ptr = (xfer->Buf) + (MAX_PACKET * (packetIdx-1));

    /* Now read in the data */
    actual_size = read(Chl, packet_ptr, xfer_size);
    if (actual_size != xfer_size) {
      printf("ERROR:  read_xfer not able to receive buffer (packet %d).\n",
     packetIdx);
      return 0;
    }
    xfer->Idx += actual_size;
    tot_size  += actual_size;   /* How many bytes so far */
  }

#ifdef GE_DEBUG
  printf(" SUCCESSFUL (%d bytes).\n", tot_size);
#endif
  xfer->Idx = tot_size;
  return tot_size;
}


#ifdef _NO_PROTO
int write_xfer(Chl, xfer)
int        Chl;
XferBuffer *xfer;
{
#else
int write_xfer(int Chl, XferBuffer *xfer) {
#endif

  int   int_size, actual_size;
  int   num_packets;
  int   buf_size;
  int   packetIdx;
  int   cur_packet_size;
  char *packet_ptr;

  int_size = sizeof(int);

  /* Calculate how many packets will be required to send buffer */
  buf_size = xfer->Idx;
  num_packets = buf_size / MAX_PACKET;   /* whole packets           */
  if (buf_size % MAX_PACKET != 0)        /* plus any partial buffer */
    num_packets++;

#ifdef GE_DEBUG
  printf("write_xfer %d bytes using %d packets...", buf_size, num_packets);
#endif

  /* first we need to tell receiver how many packets to expect */
```

```
  actual_size = write(Ch1, &num_packets, int_size);
  if (actual_size != int_size) {
    printf("ERROR:  write_xfer not able to transfer buffer (num_packets).\n");
    return 0;
  }

  /* Now send each packet to receiver */
  packet_ptr = xfer->Buf;
  for (packetIdx = 1; packetIdx <= num_packets; packetIdx++) {

    if (packetIdx == num_packets)
      cur_packet_size = buf_size % MAX_PACKET;
    else
      cur_packet_size = MAX_PACKET;

    /* Data sent in a size, data pair */
    actual_size = write(Ch1, &cur_packet_size, int_size); /* first size */
    if (actual_size != int_size) {
      printf("ERROR:  write_xfer not able to transfer buffer (packet size).\n");
      return 0;
    }

    actual_size = write(Ch1, packet_ptr, cur_packet_size); /* next data */
    if (actual_size != cur_packet_size) {
      printf("ERROR:  write_xfer not able to transfer buffer (packet %d).\n",
     packetIdx);
      return 0;
    }

    packet_ptr += MAX_PACKET;
  }
#ifdef GE_DEBUG
  printf("SUCCESSFUL\n");
#endif
  return buf_size;
}


#ifdef _NO_PROTO
BOOLEAN packString(str, xfer)
char* str;
XferBuffer *xfer;
{
#else
BOOLEAN packString(char* str, XferBuffer *xfer) {
#endif
  int delta;

  if (str != NULL) {
    delta = strlen(str) + 1;

    if (xfer_space_avail(xfer, delta)) {
      strcpy((char *) &(xfer->Buf[xfer->Idx]), str);
      xfer->Idx += delta;

      return true;
    }
    else {
#ifdef GE_DEBUG
      printf("Error in packString\n");
#endif
      return false;
    }
  }
  else {    /* If null, we still need something to occupy space */
    char* blank_line = strdup("");
    delta = strlen(blank_line) + 1;

    if (xfer_space_avail(xfer, delta)) {
      strcpy((char*) &(xfer->Buf[xfer->Idx]), blank_line);
      xfer->Idx += delta;
      free(blank_line);

      return true;
    }
    else {
      free(blank_line);

#ifdef GE_DEBUG
      printf("Error in packString\n");
#endif
      return false;
    }
  }
}


#ifdef _NO_PROTO
BOOLEAN packInteger(inInt, xfer)
int inInt;
XferBuffer *xfer;
{
#else
BOOLEAN packInteger(int inInt, XferBuffer *xfer) {
#endif
  int delta;

  delta = sizeof(int);

  if (xfer_space_avail(xfer, delta)) {
    memcpy((char*)&(xfer->Buf[xfer->Idx]), &inInt, delta);
    xfer->Idx += delta;

    return true;
  }
  else {
```

```
#ifdef GE_DEBUG
    printf("Error in packInteger\n");
#endif
    return false;
  }
}


#ifdef _NO_PROTO
BOOLEAN packBoolean(Bool, xfer)
BOOLEAN Bool;
XferBuffer *xfer;
{
#else
BOOLEAN packBoolean(BOOLEAN Bool, XferBuffer *xfer) {
#endif
  int delta;

  delta = sizeof(BOOLEAN);

  if (xfer_space_avail(xfer, delta)) {
    memcpy((char*)&(xfer->Buf[xfer->Idx]), &Bool, delta);
    xfer->Idx += delta;

    return true;
  }
  else {
#ifdef GE_DEBUG
    printf("Error in packBoolean\n");
#endif
    return false;
  }
}

#ifdef _NO_PROTO
BOOLEAN packID_LIST(idList, xfer)
ID_LIST idList;
XferBuffer *xfer;
{
#else
BOOLEAN packID_LIST(ID_LIST idList, XferBuffer *xfer) {
#endif

  ID_LIST IDptr;
  BOOLEAN packed;

  char *End_of_List = "}End_of_List";

  packed = true;
  IDptr  = idList;

  while ((IDptr != NULL) && packed) {
    packed &= packString(IDptr->id, xfer);
    IDptr = IDptr->next;
  }

  packed &= packString(End_of_List, xfer);

#ifdef GE_DEBUG
  if (!packed)
    printf("Error in packID_LIST\n");
#endif

  return packed;

}



#ifdef _NO_PROTO
BOOLEAN packSpline(spList, xfer)
SPLINE_PTR spList;
XferBuffer *xfer;
{
#else
BOOLEAN packSpline(SPLINE_PTR spList, XferBuffer *xfer) {
#endif

  SPLINE_PTR SPptr;
  BOOLEAN packed;

  char *Spline         = "}Spline";
  char *End_of_Spline = "}End_of_Spline";

  packed = true;
  SPptr  = spList;

  while ((SPptr != NULL) && packed) {
    packed &= packString(Spline, xfer);      /* Need a string at head to */
    packed &= packInteger(SPptr->x, xfer);    /* check for end of spline  */
    packed &= packInteger(SPptr->y, xfer);

    SPptr = SPptr->next;
  }

  packed &= packString(End_of_Spline, xfer);

#ifdef GE_DEBUG
  if (!packed)
    printf("Error in packSpline\n");
#endif

  return packed;

}
```

```
#ifdef _NO_PROTO
char* unpackString(xfer)
XferBuffer *xfer;
{
#else
char* unpackString(XferBuffer *xfer) {
#endif

  char *temp;

  temp = (char*) malloc(strlen((char*)&(xfer->Buf[xfer->Idx]))+1);

  strcpy(temp, (char*)&(xfer->Buf[xfer->Idx]));
  xfer->Idx += (strlen(temp)+1);

  return temp;
}
 .

#ifdef _NO_PROTO
int unpackInteger(xfer)
XferBuffer *xfer;
{
#else
int unpackInteger(XferBuffer *xfer) {
#endif

  int
      temp,
      delta;

  delta = sizeof(int);

  memcpy((char*)&temp, (char*)&(xfer->Buf[xfer->Idx]), delta);
  xfer->Idx += delta;

  return temp;
}


#ifdef _NO_PROTO
BOOLEAN unpackBoolean(xfer)
XferBuffer *xfer;
{
#else
BOOLEAN unpackBoolean(XferBuffer *xfer) {
#endif

  BOOLEAN temp;
  int     delta;

  delta = sizeof(BOOLEAN);

  memcpy((char*)&temp, (char*)&(xfer->Buf[xfer->Idx]), delta);
  xfer->Idx += delta;

  return temp;
}


#ifdef _NO_PROTO
ID_LIST unpackID_LIST(xfer)
XferBuffer *xfer;
{
#else
ID_LIST unpackID_LIST(XferBuffer *xfer) {
#endif

  BOOLEAN contLoop;
  ID_LIST IDptr, IDhead;
  char    *Id;

  char *End_of_List = "}End_of_List";

  contLoop = true;
  IDhead = NULL;

  while (contLoop) {
    Id = unpackString(xfer);

    if (strcmp(Id,(char*) End_of_List) == 0) {
      contLoop = false;
    }
    else {
      if (IDhead == NULL) {
IDptr  = (ID_LIST) malloc(sizeof(ID_NODE));
IDhead = IDptr;
      }
      else {
IDptr->next = (ID_LIST) malloc(sizeof(ID_NODE));
IDptr = IDptr->next;
      }
      IDptr->id   = Id;
      IDptr->next = NULL;
    }
  }
  free(Id);     /* Recover the End_of_List marker */
  return IDhead;

}


#ifdef _NO_PROTO
SPLINE_PTR unpackSpline(xfer)
```

```
XferBuffer *xfer;
{
#else
SPLINE_PTR unpackSpline(XferBuffer *xfer) {
#endif

  BOOLEAN contLoop;
  SPLINE_PTR SPptr, SPhead;
  char    *header;

  char *End_of_Spline = "}End_of_Spline";

  contLoop = true;
  SPhead = NULL;

  while (contLoop) {
    header = unpackString(xfer);

    if (strcmp(header,(char*) End_of_Spline) == 0) {
      contLoop = false;
    }
    else {
      if (SPhead == NULL) {
SPptr  = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
SPhead = SPptr;
      }
      else {
SPptr->next = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
SPptr = SPptr->next;
      }
      SPptr->x    = unpackInteger(xfer);
      SPptr->y    = unpackInteger(xfer);
      SPptr->next = NULL;
    }
    free(header);     /* Recover the Spline marker */
  }
  return SPhead;

}


#ifdef _NO_PROTO
BOOLEAN readChkWord(chkWord, Chl)
int *chkWord;
int Chl;
{
#else
BOOLEAN readChkWord(int *chkWord, int Chl) {
#endif

  int xfer_size, int_size;

  int_size = sizeof(int);

  if ((read(Chl, &xfer_size, int_size) == int_size) &&
      (read(Chl, chkWord, xfer_size) == xfer_size))
    return true;
  else {
#ifdef GE_DEBUG
    printf("Error in readChkWord\n");
#endif
    return false;
  }
}


#ifdef _NO_PROTO
BOOLEAN writeChkWord(chkWord, Chl)
int chkWord;
int Chl;
{
#else
BOOLEAN writeChkWord(int chkWord, int Chl) {
#endif

  int int_size;

  int_size = sizeof(int);

  if ((write(Chl, &int_size, int_size) == int_size) &&
      (write(Chl, &chkWord, int_size) == int_size))
    return true;
  else {
#ifdef GE_DEBUG
    printf("Error in writeChkWord\n");
#endif
    return false;
  }
}


#ifdef _NO_PROTO
BOOLEAN synch_write(Chl)
int Chl;
{
#else
BOOLEAN synch_write(int Chl) {
#endif

  return writeChkWord(CHKWORD, Chl);
}


#ifdef _NO_PROTO
BOOLEAN synch_read(Chl)
int Chl;
```

```
{
#else
BOOLEAN synch_read(int Chl) {
#endif

  int chkWord;

  return readChkWord(&chkWord, Chl);
}


#ifdef _NO_PROTO
BOOLEAN packGraphNames(graph, xfer)
GRAPH_DESC graph;
XferBuffer *xfer;
{
#else
BOOLEAN packGraphNames(GRAPH_DESC graph, XferBuffer *xfer) {
#endif

  BOOLEAN packed;
  packed = true;

  packed &= packString(graph->root_op_name, xfer);
  packed &= packInteger(graph->root_op_num, xfer);
  packed &= packString(graph->parent_op_name, xfer);
  packed &= packInteger(graph->parent_op_num, xfer);
  packed &= packString(graph->current_op_name, xfer);
  packed &= packInteger(graph->current_op_num, xfer);
  packed &= packInteger(graph->cur_op_spec_met, xfer);
  packed &= packInteger(graph->cur_op_spec_met_unit, xfer);
  packed &= packInteger(graph->cur_op_is_terminator, xfer);

#ifdef GE_DEBUG
  if (!packed)
    printf("error in packGraphNames\n");
#endif

  return packed;
}


#ifdef _NO_PROTO
void unpackGraphNames(graph, xfer)
GRAPH_DESC graph;
XferBuffer *xfer;
{
#else
void unpackGraphNames(GRAPH_DESC graph, XferBuffer *xfer) {
#endif

  graph->root_op_name         = unpackString(xfer);
  graph->root_op_num          = unpackInteger(xfer);
  graph->parent_op_name       = unpackString(xfer);
  graph->parent_op_num        = unpackInteger(xfer);
  graph->current_op_name      = unpackString(xfer);
  graph->current_op_num       = unpackInteger(xfer);
  graph->cur_op_spec_met      = unpackInteger(xfer);
  graph->cur_op_spec_met_unit = unpackInteger(xfer);
  graph->cur_op_is_terminator = unpackInteger(xfer);

}


#ifdef _NO_PROTO
BOOLEAN packOperatorList(operator_list, xfer)
OP_LIST operator_list;
XferBuffer *xfer;
{
#else
BOOLEAN packOperatorList(OP_LIST operator_list, XferBuffer *xfer) {
#endif

  BOOLEAN packed;
  OP_LIST OPlist;
  OPERATOR OP;

  char *Operator         = "}Operator";
  char *End_of_Operators = "}End_of_Operators";

  packed = true;
  OPlist = operator_list;

  while ((OPlist != NULL) && packed) {
    OP = OPlist->op;

    packed &= packString(Operator, xfer);
    packed &= packInteger(OP->id, xfer);
    packed &= packInteger(OP->op_num, xfer);
    packed &= packInteger(OP->x, xfer);
    packed &= packInteger(OP->y, xfer);
    packed &= packInteger(OP->radius, xfer);
    packed &= packInteger(OP->color, xfer);
    packed &= packString(OP->label, xfer);
    packed &= packInteger(OP->label_font, xfer);
    packed &= packInteger(OP->label_x_offset, xfer);
    packed &= packInteger(OP->label_y_offset, xfer);
    packed &= packInteger(OP->timing_type, xfer);
    packed &= packInteger(OP->met, xfer);
    packed &= packInteger(OP->met_unit, xfer);
    packed &= packInteger(OP->met_font, xfer);
    packed &= packInteger(OP->met_x_offset, xfer);
    packed &= packInteger(OP->met_y_offset, xfer);
    packed &= packID_LIST(OP->met_reqmts, xfer);
    packed &= packInteger(OP->period, xfer);
```

```
    packed &= packInteger(OP->period_unit, xfer);
    packed &= packID_LIST(OP->period_reqmts, xfer);
    packed &= packInteger(OP->fw, xfer);
    packed &= packInteger(OP->fw_unit, xfer);
    packed &= packID_LIST(OP->fw_reqmts, xfer);
    packed &= packInteger(OP->mrt, xfer);
    packed &= packInteger(OP->mrt_unit, xfer);
    packed &= packID_LIST(OP->mrt_reqmts, xfer);
    packed &= packInteger(OP->mcp, xfer);
    packed &= packInteger(OP->mcp_unit, xfer);
    packed &= packID_LIST(OP->mcp_reqmts, xfer);
    packed &= packInteger(OP->trigger_type, xfer);
    packed &= packID_LIST(OP->trigger_set, xfer);
    packed &= packString(OP->if_condition, xfer);
    packed &= packID_LIST(OP->trigger_reqmts, xfer);
    packed &= packString(OP->output_guard_list, xfer);
    packed &= packString(OP->exception_list, xfer);
    packed &= packString(OP->timer_op_list, xfer);
    packed &= packID_LIST(OP->key_word_list, xfer);
    packed &= packString(OP->operator_informal_desc, xfer);
    packed &= packString(OP->operator_formal_desc, xfer);
    packed &= packString(OP->operator_impl_lang, xfer);
    packed &= packBoolean(OP->is_composite, xfer);
    packed &= packBoolean(OP->is_terminator, xfer);
    packed &= packBoolean(OP->is_new, xfer);
    packed &= packBoolean(OP->is_modified, xfer);
    packed &= packBoolean(OP->is_deleted, xfer);

    OPlist = OPlist->next;
  }

  packed &= packString(End_of_Operators, xfer);

#ifdef GE_DEBUG
  if (!packed)
    printf("error in packOperatorList\n");
#endif

  return packed;

}


#ifdef _NO_PROTO
OP_LIST unpackOperatorList(xfer)
XferBuffer *xfer;
{
#else
OP_LIST unpackOperatorList(XferBuffer *xfer) {
#endif

  BOOLEAN contLoop;
  OP_LIST OPptr, OPhead;
  char    *header;
  OPERATOR OP;

  char *End_of_Operators = "}End_of_Operators";

  contLoop = true;
  OPhead = NULL;

  while (contLoop) {
    header = unpackString(xfer);

    if (strcmp(header,(char*) End_of_Operators) == 0) {
      contLoop = false;
    }
    else {
      if (OPhead == NULL) {
OPptr  = (OP_LIST) malloc(sizeof(OP_LIST_NODE));
OPhead = OPptr;
      }
      else {
OPptr->next = (OP_LIST) malloc(sizeof(OP_LIST_NODE));
OPptr = OPptr->next;
      }
      OP = (OPERATOR) malloc(sizeof(OP_NODE));
      OPptr->op = OP;

      OP->id                    = unpackInteger(xfer);
      OP->op_num                = unpackInteger(xfer);
      OP->x                     = unpackInteger(xfer);
      OP->y                     = unpackInteger(xfer);
      OP->radius                = unpackInteger(xfer);
      OP->color                 = unpackInteger(xfer);
      OP->label                 = unpackString(xfer);
      OP->label_font            = unpackInteger(xfer);
      OP->label_x_offset        = unpackInteger(xfer);
      OP->label_y_offset        = unpackInteger(xfer);
      OP->timing_type           = unpackInteger(xfer);
      OP->met                   = unpackInteger(xfer);
      OP->met_unit              = unpackInteger(xfer);
      OP->met_font              = unpackInteger(xfer);
      OP->met_x_offset          = unpackInteger(xfer);
      OP->met_y_offset          = unpackInteger(xfer);
      OP->met_reqmts            = unpackID_LIST(xfer);
      OP->period                = unpackInteger(xfer);
      OP->period_unit           = unpackInteger(xfer);
      OP->period_reqmts         = unpackID_LIST(xfer);
      OP->fw                    = unpackInteger(xfer);
      OP->fw_unit               = unpackInteger(xfer);
      OP->fw_reqmts             = unpackID_LIST(xfer);
      OP->mrt                   = unpackInteger(xfer);
      OP->mrt_unit              = unpackInteger(xfer);
      OP->mrt_reqmts            = unpackID_LIST(xfer);
      OP->mcp                   = unpackInteger(xfer);
```

```
      OP->mcp_unit               = unpackInteger(xfer);
      OP->mcp_reqmts             = unpackID_LIST(xfer);
      OP->trigger_type           = unpackInteger(xfer);
      OP->trigger_set            = unpackID_LIST(xfer);
      OP->if_condition           = unpackString(xfer);
      OP->trigger_reqmts         = unpackID_LIST(xfer);
      OP->output_guard_list      = unpackString(xfer);
      OP->exception_list         = unpackString(xfer);
      OP->timer_op_list          = unpackString(xfer);
      OP->key_word_list          = unpackID_LIST(xfer);
      OP->operator_informal_desc = unpackString(xfer);
      OP->operator_formal_desc   = unpackString(xfer);
      OP->operator_impl_lang     = unpackString(xfer);
      OP->is_composite           = unpackBoolean(xfer);
      OP->is_terminator          = unpackBoolean(xfer);
      OP->is_new                 = unpackBoolean(xfer);
      OP->is_modified            = unpackBoolean(xfer);
      OP->is_deleted             = unpackBoolean(xfer);

      OPptr->next = NULL;
    }
    free(header);     /* Recover the header marker */
  }
  return OPhead;

}


#ifdef _NO_PROTO
BOOLEAN packStreamList(stream_list, xfer)
ST_LIST stream_list;
XferBuffer *xfer;
{
#else
BOOLEAN packStreamList(ST_LIST stream_list, XferBuffer *xfer) {
#endif

  BOOLEAN packed;
  ST_LIST STlist;
  STREAM ST;

  char *Stream          = "}Stream";
  char *End_of_Streams = "}End_of_Streams";

  packed = true;
  STlist = stream_list;

  while ((STlist != NULL) && packed) {
    ST = STlist->st;

    packed &= packString(Stream, xfer);
    packed &= packInteger(ST->id, xfer);
    packed &= packString(ST->label, xfer);
    packed &= packInteger(ST->label_font, xfer);
    packed &= packInteger(ST->label_x_offset, xfer);
    packed &= packInteger(ST->label_y_offset, xfer);
    packed &= packInteger(ST->from, xfer);
    packed &= packInteger(ST->to, xfer);
    packed &= packSpline(ST->arc, xfer);
    packed &= packInteger(ST->latency, xfer);
    packed &= packInteger(ST->latency_unit, xfer);
    packed &= packInteger(ST->latency_font, xfer);
    packed &= packInteger(ST->latency_x_offset, xfer);
    packed &= packInteger(ST->latency_y_offset, xfer);
    packed &= packString(ST->stream_type_name, xfer);
    packed &= packString(ST->state_initial_value, xfer);
    packed &= packBoolean(ST->is_state_variable, xfer);
    packed &= packBoolean(ST->is_new, xfer);
    packed &= packBoolean(ST->is_modified, xfer);
    packed &= packBoolean(ST->is_deleted, xfer);

    STlist = STlist->next;
  }

  packed &= packString(End_of_Streams, xfer);

#ifdef GE_DEBUG
  if (!packed)
    printf("error in packStreamList\n");
#endif

  return packed;

}


#ifdef _NO_PROTO
ST_LIST unpackStreamList(xfer)
XferBuffer *xfer;
{
#else
ST_LIST unpackStreamList(XferBuffer *xfer) {
#endif

  BOOLEAN contLoop;
  ST_LIST STptr, SThead;
  char    *header;
  STREAM  ST;

  char *End_of_Streams = "}End_of_Streams";

  contLoop = true;
  SThead = NULL;

  while (contLoop) {
```

```
    header = unpackString(xfer);

    if (strcmp(header,(char*) End_of_Streams) == 0) {
      contLoop = false;
    }
    else {
      if (SThead == NULL) {
STptr  = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
SThead = STptr;
      }
      else {
STptr->next = (ST_LIST) malloc(sizeof(ST_LIST_NODE));
STptr = STptr->next;
      }
      ST = (STREAM) malloc(sizeof(ST_NODE));
      STptr->st = ST;

      ST->id                  = unpackInteger(xfer);
      ST->label               = unpackString(xfer);
      ST->label_font          = unpackInteger(xfer);
      ST->label_x_offset      = unpackInteger(xfer);
      ST->label_y_offset      = unpackInteger(xfer);
      ST->from                = unpackInteger(xfer);
      ST->to                  = unpackInteger(xfer);
      ST->arc                 = unpackSpline(xfer);
      ST->latency             = unpackInteger(xfer);
      ST->latency_unit        = unpackInteger(xfer);
      ST->latency_font        = unpackInteger(xfer);
      ST->latency_x_offset    = unpackInteger(xfer);
      ST->latency_y_offset    = unpackInteger(xfer);
      ST->stream_type_name    = unpackString(xfer);
      ST->state_initial_value = unpackString(xfer);
      ST->is_state_variable   = unpackBoolean(xfer);
      ST->is_new              = unpackBoolean(xfer);
      ST->is_modified         = unpackBoolean(xfer);
      ST->is_deleted          = unpackBoolean(xfer);

      STptr->next = NULL;
    }
    free(header);     /* Recover the header marker */
  }
  return SThead;

}


#ifdef _NO_PROTO
BOOLEAN readAction(next_action, xfer, Chl)
ACTION next_action;
XferBuffer *xfer;
int Chl;
{
#else
BOOLEAN readAction(ACTION next_action, XferBuffer *xfer, int Chl) {
#endif

  int chkWord;

  /* recover any allocated space to next_action */
  free((char*) next_action->next_op); next_action->next_op = NULL;

  /* fetch data */
  if (read_xfer(Chl, xfer)) {

    xfer->Idx = 0;
    /* unpack next_action */
    next_action->option      = (GE_ACTION_TYPE) unpackInteger(xfer);
    next_action->reinvoke    = unpackBoolean(xfer);
    next_action->next_op     = unpackString(xfer);
    next_action->next_op_num = unpackInteger(xfer);
    chkWord = unpackInteger(xfer);

    if (chkWord = CHKWORD)
      return true;
    else {
#ifdef GE_DEBUG
      printf("error in readAction\n");
#endif
      return false;
    }
  }
  else {
#ifdef GE_DEBUG
    printf("error in readAction\n");
#endif
    return false;
  }

}


#ifdef _NO_PROTO
BOOLEAN writeAction(next_action, xfer, Chl)
ACTION next_action;
XferBuffer *xfer;
int Chl;
{
#else
BOOLEAN writeAction(ACTION next_action, XferBuffer *xfer, int Chl) {
#endif

  BOOLEAN packed;

  int int_size;

  packed = true;
```

```
  int_size = sizeof(int);

  xfer->Idx = 0;

  /* build the buffer */

  packed &= packInteger((int) next_action->option, xfer);
  packed &= packBoolean(next_action->reinvoke, xfer);
  packed &= packString(next_action->next_op, xfer);
  packed &= packInteger(next_action->next_op_num, xfer);
  packed &= packInteger(CHKWORD, xfer);

  /* send data */
  if (packed && write_xfer(Chl, xfer))
    return true;
  else {
#ifdef GE_DEBUG
    printf("error in writeAction\n");
#endif
    return false;
  }
}


#ifdef _NO_PROTO
BOOLEAN readGraphDesc(graph, xfer, Chl)
GRAPH_DESC graph;
XferBuffer *xfer;
int Chl;
{
#else
BOOLEAN readGraphDesc(GRAPH_DESC graph, XferBuffer *xfer, int Chl) {
#endif

  int chkWord;

  /* recover any allocated space to graph */
  graph_release(graph);

  /* fetch data */
  if (read_xfer(Chl, xfer)) {

    xfer->Idx = 0;
    /* unpack next_action */
    unpackGraphNames(graph, xfer);
    graph->cur_op_spec         = unpackString(xfer);
    graph->input_list          = unpackStreamList(xfer);
    graph->output_list         = unpackStreamList(xfer);
    graph->operator_list       = unpackOperatorList(xfer);
    graph->stream_list         = unpackStreamList(xfer);
    graph->timer_list          = unpackID_LIST(xfer);
    graph->graph_informal_desc = unpackString(xfer);
    graph->avail_impl_langs    = unpackID_LIST(xfer);
    graph->global_types        = unpackString(xfer);

    if (unpackInteger(xfer) == CHKWORD)
      return true;
    else {
#ifdef GE_DEBUG
      printf("****ERROR IN readGraphDesc unpacking graph desc****\n");
#endif
      return false;
    }
  }
  else {
#ifdef GE_DEBUG
      printf("****ERROR IN readGraphDesc reading xfer buffer****\n");
#endif
    return false;
  }
}



#ifdef _NO_PROTO
BOOLEAN writeGraphDesc(graph, xfer, Chl)
GRAPH_DESC graph;
XferBuffer *xfer;
int Chl;
{
#else
BOOLEAN writeGraphDesc(GRAPH_DESC graph, XferBuffer *xfer, int Chl) {
#endif

  BOOLEAN packed;
  int int_size;

  packed = true;
  int_size = sizeof(int);

  xfer->Idx = 0;
  /* build the buffer */
  packed &= packGraphNames(graph, xfer);
  packed &= packString(graph->cur_op_spec, xfer);
  packed &= packStreamList(graph->input_list, xfer);
  packed &= packStreamList(graph->output_list, xfer);
  packed &= packOperatorList(graph->operator_list, xfer);
  packed &= packStreamList(graph->stream_list, xfer);
  packed &= packID_LIST(graph->timer_list, xfer);
  packed &= packString(graph->graph_informal_desc, xfer);
  packed &= packID_LIST(graph->avail_impl_langs, xfer);
  packed &= packString(graph->global_types, xfer);
  packed &= packInteger(CHKWORD, xfer);

  /* send data */
  if (packed && write_xfer(Chl, xfer))
```

```
    return true;
  else {
#ifdef GE_DEBUG
    printf("error in writeGraphDesc\n");
#endif
    return false;
  }
}


#ifdef _NO_PROTO
BOOLEAN readErrorMsgs(errs, xfer, Chl)
ERROR_MSGS *errs;
XferBuffer *xfer;
int Chl;
{
#else
BOOLEAN readErrorMsgs(ERROR_MSGS *errs, XferBuffer *xfer, int Chl) {
#endif

  ERROR_MSGS ErrPtr;

  int int_size, xfer_size;

  int_size = sizeof(int);

  if (xfer_size = read_xfer(Chl, xfer)) {

    /* Recover allocated memory */
    err_msgs_release(errs);

    xfer->Idx = 0;
    xfer_size -= int_size;    /* Stop before the CHKWORD */

    while ((xfer->Idx) < xfer_size) {
      if (*errs == NULL) {
ErrPtr = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));
*errs = ErrPtr;
      }
      else {
ErrPtr->next = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));
ErrPtr = ErrPtr->next;
      }
      ErrPtr->parent_op_num   = unpackInteger(xfer);
      ErrPtr->parent_op_label = unpackString(xfer);
      ErrPtr->cur_op_num      = unpackInteger(xfer);
      ErrPtr->cur_op_label    = unpackString(xfer);
      ErrPtr->msg             = unpackString(xfer);
      ErrPtr->next            = NULL;
    }

    if (unpackInteger(xfer) == CHKWORD)
      return true;
    else {
#ifdef GE_DEBUG
      printf("error in readErrorMsgs\n");
#endif
      return false;
    }
  }
  else {
#ifdef GE_DEBUG
    printf("error in readErrorMsgs\n");
#endif
    return false;
  }
}


#ifdef _NO_PROTO
BOOLEAN writeErrorMsgs(errs, xfer, Chl)
ERROR_MSGS errs;
XferBuffer *xfer;
int Chl;
{
#else
BOOLEAN writeErrorMsgs(ERROR_MSGS errs, XferBuffer *xfer, int Chl) {
#endif

  ERROR_MSGS ErrPtr;
  BOOLEAN packed;

  packed = true;
  ErrPtr = errs;

  xfer->Idx = 0;
  while ((ErrPtr != NULL) && packed) {
    packed &= packInteger(ErrPtr->parent_op_num, xfer);
    packed &= packString(ErrPtr->parent_op_label, xfer);
    packed &= packInteger(ErrPtr->cur_op_num, xfer);
    packed &= packString(ErrPtr->cur_op_label, xfer);
    packed &= packString(ErrPtr->msg, xfer);

    ErrPtr = ErrPtr->next;
  }

  packed &= packInteger(CHKWORD, xfer);

  if (packed && write_xfer(Chl, xfer))
    return true;
  else {
#ifdef GE_DEBUG
    printf("error in writeErrorMsgs\n");
#endif
    return false;
  }
```

```
#endif

#ifdef __cplusplus
}
#endif
```

```
/* ************************************************************

Name:          operator_object.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 17 Sep 92
Remarks:   This is the specification for the OperatorObject
         class.

           The OperatorObject class is the graphical
         representation of a PSDL operator.  It has three
         physical forms: circular for an atomic operator,
         two concentric circles for a non-atomic operator,
         and rectangular, for terminators.  A terminator
         simulates interaction with objects outside the system.

Reengineering:
Modified by Doug Lange to add new operator properties 9/8/96

History:

  @1   96/10/04  Ken Moeller
       Removed un-needed routines to read and write to disk.

  @2   96/10/05  Ken Moeller
       Switched over to units being passed instead of using
       a signed value of met.

  @3   96/10/06  Ken Moeller
       Removed un-needed routines to read and write to property.

  @4   96/10/07  Ken Moeller
       Changes to ge_interface.h

************************************************************ */
#ifndef operator_object_h
#define operator_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include "ge_defs.h"
#include "graph_object.h"
#include "ge_interface.h" //Added by DL 9/8/96
#include "ge_utilities.h" //Added by DL 9/13/96

class OperatorObject : public GraphObject {
protected:
  char     *_met_string_ptr;
  int      _name_width, _name_height;
  int      _met_width,  _met_height;
  BOOLEAN  _handle_selected,
           _is_selected,
           _name_selected,
           _met_selected,
           _op_handles_drawn,
           _name_handles_drawn,
           _met_handles_drawn;

//------------------- ge_interface.h --------------------------

  OP_ID   _id;                  /* vertex number */
  OP_ID   _op_num;

  int     _x,
          _y,
          _radius,
          _color;

  char   *_name_ptr;           /*  *label; */
  int     _name_font,
          _name_x,
          _name_y;

  int     _timing_type;

  int     _met,
          _met_unit,
          _met_font,
          _met_x,
          _met_y;
  ID_LIST _met_reqmts;

  int     _period,
          _period_unit;
  ID_LIST _period_reqmts;

  int     _fw,
          _fw_unit;
  ID_LIST _fw_reqmts;

  int     _mrt,
          _mrt_unit;
  ID_LIST _mrt_reqmts;

  int     _mcp,
          _mcp_unit;
  ID_LIST _mcp_reqmts;

  int     _trigger_type;
  ID_LIST _trigger_set;
  char*   _trigger_if_condition;
  ID_LIST _trigger_reqmts;

  char*   _output_guard_list;
  char*   _exception_list;
  char*   _timer_op_list;
```

```
  ID_LIST _key_word_list;
  char*   _operator_informal_desc;
  char*   _operator_formal_desc;
  char*   _operator_impl_lang;

  BOOLEAN _is_composite,
          _is_terminator;
  BOOLEAN _is_new,
          _is_modified,
          _is_deleted;

//-------------------------------------------------------------


  void set_object_font(int font_id);
  void set_text_dimensions();
  void draw_handles(GC draw_context, int x1, int y1, int x2, int y2);
  void reset_handles_drawn_state();

public:

  OperatorObject(); // Constructor which initializes all data elemets
  OperatorObject(OPERATOR op);
                   // Deep copy constructor from an OPERATOR (ge_interface)
  OperatorObject(char *in_name_ptr, OP_ID in_id, OP_ID in_op,
 int in_met, int in_unit,
                int in_x, int in_y, int in_radius, int in_color,
                BOOLEAN in_is_new, BOOLEAN in_is_composite,
                BOOLEAN in_is_terminator);
  virtual ~OperatorObject();            // Recovers all dynamic memory

  OperatorObject& operator=(OperatorObject& src);
  OPERATOR clone();             // Allocates storage and copies to OPERATOR
  void release();
  void initialize();
  void copy(OperatorObject *src);
  void read_from(OPERATOR op);
  void write_to(OPERATOR on);

  CLASS_DEF is_a() {return OPERATOROBJECT;}

  void draw(DRAW_STYLE style);
  void draw_text(DRAW_STYLE style);
  void set_default_text_location();
  void move_text(int x, int y);

  void select();
  void unselect();

  void erase();
  void erase_text();
```

```
  BOOLEAN hit(int x, int y);
  BOOLEAN over(int x, int y); // Added 8/26/96, dha

  XYPAIR center();
  XYPAIR intercept(int x, int y);

  void set_location(int x, int y);
  void move(int x, int y);
  void move_notify(CLASS_DEF , OP_ID ) {}

  void delete_notify(CLASS_DEF class_type, OP_ID deleted_obj_id);
  void undelete_notify(CLASS_DEF class_type, OP_ID deleted_obj_id);

  BOOLEAN hit_handle(int x, int y);
  void move_handle(int x, int y);
  BOOLEAN text_selected() {return (_name_selected || _met_selected);}

  int text_width();
  int text_height();
  void text_locate(int x, int y);

  void write_block(GC draw_context, int x, int y,
                   char *instring, int block_height);

  int handle_selected() {return _handle_selected;}
  void set_handle_selected(int handle);

  void set_default_name_location();
  void set_default_met_location();

  //Member functions added by DL 9/8/96; KBM 10/24/96

//------------------- Methods to get values of ge_interface.h ---------------

  OP_ID   id()                {return _id;}
  OP_ID   op_num()            {return _op_num;}

  int     x()                 {return _x;}
  int     y()                 {return _y;}
  int     radius()            {return _radius;}
  int     color()             {return _color;}

  char    *name()             {return dup_str(_name_ptr);}
  int     name_font()         {return _name_font;}
  int     name_x()            {return _name_x;}
  int     name_y()            {return _name_y;}

  int   timing_type()       {return _timing_type;}

  int   met()               {return _met;}
  int   met_unit()          {return _met_unit;}
  int   met_font()          {return _met_font;}
  int   met_x()             {return _met_x;}
```

```
int   met_y()              {return _met_y;}
ID_LIST* met_req_by_adr()     {return &_met_reqmts;}
ID_LIST met_reqmts_copy()    {return id_list_copy(_met_reqmts);}
BOOLEAN met_reqmts_avail()   {return (_met_reqmts != NULL);}

int     period()             {return _period;}
int   period_unit()        {return _period_unit;}
ID_LIST* period_req_by_adr()  {return &_period_reqmts;}
ID_LIST period_reqmts_copy()  {return id_list_copy(_period_reqmts);}
BOOLEAN period_reqmts_avail() {return (_period_reqmts != NULL);}

int   fw()                 {return _fw;}
int   fw_unit()            {return _fw_unit;}
ID_LIST* fw_req_by_adr()      {return &_fw_reqmts;}
ID_LIST fw_reqmts_copy()     {return id_list_copy(_fw_reqmts);}
BOOLEAN fw_reqmts_avail()    {return (_fw_reqmts != NULL);}

int   mrt()                {return _mrt;}
int   mrt_unit()           {return _mrt_unit;}
ID_LIST* mrt_req_by_adr()     {return &_mrt_reqmts;}
ID_LIST mrt_reqmts_copy()    {return id_list_copy(_mrt_reqmts);}
BOOLEAN mrt_reqmts_avail()   {return (_mrt_reqmts != NULL);}

int    mcp()                {return _mcp;}
int   mcp_unit()           {return _mcp_unit;}
ID_LIST* mcp_req_by_adr()     {return &_mcp_reqmts;}
ID_LIST mcp_reqmts_copy()    {return id_list_copy(_mcp_reqmts);}
BOOLEAN mcp_reqmts_avail()   {return (_mcp_reqmts != NULL);}

int   trigger_type()       {return _trigger_type;}
ID_LIST* trigger_set_adr()    {return &_trigger_set;}
ID_LIST trigger_set_copy()  {return id_list_copy(_trigger_set);}  // @4
char*   trigger_if_condition() {return dup_str(_trigger_if_condition);}
ID_LIST* trigger_req_by_adr()   {return &_trigger_reqmts;}
ID_LIST trigger_reqmts_copy()  {return id_list_copy(_trigger_reqmts);}
BOOLEAN trigger_reqmts_avail() {return (_trigger_reqmts != NULL);}

char*   output_guard_list()  {return dup_str(_output_guard_list);}
char*   exception_list()     {return dup_str(_exception_list);}
char*   timer_op_list()      {return dup_str(_timer_op_list);}

ID_LIST* key_word_list_adr()      {return &_key_word_list;}
ID_LIST key_word_list_copy()     {return id_list_copy(_key_word_list);}
char*   operator_informal_desc() {return dup_str(_operator_informal_desc);}
char**  operator_informal_desc_adr() {return &_operator_informal_desc;}
char*   operator_formal_desc() {return dup_str(_operator_formal_desc);}
char**  operator_formal_desc_adr() {return &_operator_formal_desc;}

char*   operator_impl_lang() {return dup_str(_operator_impl_lang);}
char**  operator_impl_lang_adr() {return &_operator_impl_lang;}

BOOLEAN is_composite()       {return _is_composite;}
BOOLEAN is_terminator()      {return _is_terminator;}
BOOLEAN is_new()             {return _is_new;}
BOOLEAN is_modified()        {return _is_modified;}
BOOLEAN is_deleted()         {return _is_deleted;}

//------------------- Methods to set values of ge_interface.h ---------------


void id(OP_ID id)            {_id = id;}
void op_num(OP_ID op_num)    {_op_num = op_num;}

void x(int x)                {_x = x;}
void y(int y)                {_y = y;}
void radius(int radius)      {_radius = radius;}
void color(int color)        {_color = color;}


void name(char *new_name);
void name_font(int name_font) {_name_font = name_font;}
void name_x(int name_x)       {_name_x = name_x;}
void name_y(int name_y)       {_name_y = name_y;}

void timing_type(int timing_type) {_timing_type = timing_type;}

void met(int met)            {_met = met;}
void met_unit(int met_unit)  {_met_unit = met_unit;}
void met_font(int met_font)  {_met_font = met_font;}
void met_x(int met_x)        {_met_x = met_x;}
void met_y(int met_y)        {_met_y = met_y;}
void met_reqmts_dup(ID_LIST x)   {id_list_replace(&_met_reqmts, x);}
void set_met(int met, int unit);

void period(int period)      {_period = period;}
void period_unit(int period_unit) {_period_unit = period_unit;}
void period_reqmts_dup(ID_LIST x) {id_list_replace(&_period_reqmts, x);}

void fw(int fw)              {_fw = fw;}
void fw_unit(int fw_unit)    {_fw_unit = fw_unit;}
void fw_reqmts_dup(ID_LIST x)    {id_list_replace(&_fw_reqmts, x);}

void mrt(int mrt)            {_mrt = mrt;}
void mrt_unit(int mrt_unit)  {_mrt_unit = mrt_unit;}
void mrt_reqmts_dup(ID_LIST x)   {id_list_replace(&_mrt_reqmts, x);}

void mcp(int mcp)            {_mcp = mcp;}
void mcp_unit(int mcp_unit)  {_mcp_unit = mcp_unit;}
void mcp_reqmts_dup(ID_LIST x)   {id_list_replace(&_mcp_reqmts, x);}

void trigger_type(int trigger_type) {_trigger_type = trigger_type;}
void trigger_set_dup(ID_LIST x)     {id_list_replace(&_trigger_set, x);}//@4
void trigger_reqmts_dup(ID_LIST x)  {id_list_replace(&_trigger_reqmts, x);}
void trigger_if_condition(char *x)
     {free(_trigger_if_condition);   _trigger_if_condition = dup_str(x);}
```

```
void output_guard_list(char *x)
     {free(_output_guard_list);      _output_guard_list = dup_str(x);}
void exception_list(char *x)
     {free(_exception_list);         _exception_list = dup_str(x);}
void timer_op_list(char *x)
     {free(_timer_op_list);          _timer_op_list = dup_str(x);}

void key_word_list_dup(ID_LIST x) {id_list_replace(&_key_word_list, x);}
void operator_informal_desc(char *x)
     {free(_operator_informal_desc); _operator_informal_desc = dup_str(x);}
void operator_formal_desc(char *x)
     {free(_operator_formal_desc);   _operator_formal_desc = dup_str(x);}

void operator_impl_lang(char *x)
     {free(_operator_impl_lang);     _operator_impl_lang = dup_str(x);}
```

```
  void is_composite(BOOLEAN is_composite)   {_is_composite = is_composite;}
  void is_terminator(BOOLEAN is_terminator);

  void is_new(BOOLEAN is_new)               {_is_new = is_new;}
  void is_modified(BOOLEAN is_modified)     {_is_modified = is_modified;}
  void set_modified()                       {_is_modified = true;}
  void is_deleted(BOOLEAN is_deleted)       {_is_deleted = is_deleted;}
  void set_deleted()                        {_is_deleted = true;}

};

#endif;
```

```
/* ************************************************************

Name:          operator_object.C
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 17 Sep 92
Remarks:   This is the implementation of the OperatorObject
         class.

           The OperatorObject class is the graphical
         representation of a PSDL operator.  It has three
         physical forms: circular for an atomic operator,
         two concentric circles for a non-atomic operator,
         and rectangular, for terminators.  A terminator
         simulates interaction with objects outside the system.


Credits:    Portions of code are adapted from the following:

          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.

Reengineering:
Modified by Doug Lange to add member functions for new
operator properties.  9/8/96

History:

  @1   96/10/04  Ken Moeller
       Removed code to read and write to disk.

  @2   96/10/05  Ken Moeller
       Corrections to building oprator from ge_interface.

  @3   96/10/06  Ken Moeller
       Removed code to read and write to property.

  @4   96/10/06  Ken Moeller
       Fixes to time units.

  @5   96/10/07  Ken Moeller
       Changes to ge_interface.h

  @6   96/10/08  Ken Moeller
       Switched to dup_str which produces an empty string ("")
       for a NULL pointer.

************************************************************ */
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <stream.h>
#include "operator_object.h"
#include "ge_utilities.h"                          // @6

#define COMPOSITE_SPACE 5
#define NUM_STRING_LENGTH 15
#define NONE 0
#define UPPERLEFT  1
#define UPPERRIGHT 2
#define LOWERRIGHT 3
#define LOWERLEFT  4
#define MINRADIUS  5


OperatorObject::OperatorObject() : GraphObject() {

  initialize();

}


// Added by Doug Lange 9/10/96.  Builds operator from SDE structures

OperatorObject::OperatorObject(OPERATOR op) : GraphObject() {

  initialize();
  this->read_from(op);

}

OperatorObject::OperatorObject(char *in_name_ptr,
       OP_ID in_id, OP_ID in_op,
                              int in_met, int in_met_unit,
       int in_x, int in_y,
                              int in_radius, int in_color,
                              BOOLEAN in_is_new,
                              BOOLEAN in_is_composite,
                              BOOLEAN in_is_terminator) {

  initialize();
  _name_ptr           = dup_str(in_name_ptr);
  _id           = in_id;
  _op_num                 = in_op;
  _x           = in_x;
  _y           = in_y;
  _radius           = in_radius;
```

```
  _color           = in_color;
  set_met(in_met, in_met_unit);
  is_terminator(in_is_terminator);              // over-rides _met values
  _is_composite           = in_is_composite;
  _is_new                 = in_is_new;


  set_text_dimensions();
  set_default_text_location();
  set_default_met_location();
  reset_handles_drawn_state();
}

OperatorObject::~OperatorObject() {

#ifdef GE_DEBUG
//  printf("OperatorObject Destructor for: %s\n", _name_ptr);
#endif

  release();  // Recover all dynamic memory

}

OperatorObject& OperatorObject::operator=(OperatorObject& src) {

  release();
  copy(&src);

  return *this;
}


OPERATOR OperatorObject::clone() {

  OPERATOR on = (OPERATOR) malloc(sizeof(OP_NODE));

  write_to(on);

  return on;

}


//   Draws handles around the given coordinates.  Drawn in
// exclusive-or mode to simplify erasure.

void OperatorObject::draw_handles(GC draw_context, int x1,
                                  int y1, int x2, int y2) {

  XSetFunction(_display_ptr, draw_context, GXxor);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y2 - HANDLESIZE, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y1, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y2 - HANDLESIZE,
                 HANDLESIZE, HANDLESIZE);
  XSetFunction(_display_ptr, draw_context, GXcopy);
}

//   Determines the dimensions of the text blocks.  _name_font
// and _met_font must be set before calling.  Text strings
// are automatically broken at underscores, except for
// underscores in the first and last character of the string.

void OperatorObject::set_text_dimensions() {
  int i, end_prev_index = -1, widest_string = 0, num_breaks = 0,
    widest_start_index, temp_width, temp_height, str_len;
  char temp_str[INPUT_LINE_SIZE];

  if(_name_ptr == NULL) {
    _name_width = 0;
    _name_height = 0;
  }
  else {
    str_len = strlen(_name_ptr);
    for(i = 0; i < str_len; i++)
      if(_name_ptr[i] == '_') {
        if((i != 0) && (i != (str_len - 1))) {
          num_breaks++;
          temp_width = i - end_prev_index;
          if(temp_width > widest_string) {
            widest_string = temp_width;
            widest_start_index = end_prev_index + 1;
          }
          end_prev_index = i;
        }
      }
    if(end_prev_index < (str_len - 1)) {
      temp_width = str_len - 1 - end_prev_index;
      if(temp_width > widest_string) {
        widest_string = temp_width;
        widest_start_index = end_prev_index + 1;
```

```
      }
    }
    if(widest_string == 0) {
      widest_start_index = 0;
      widest_string = str_len;
    }
    strncpy(temp_str, &(_name_ptr[widest_start_index]),
            widest_string);
    temp_str[widest_string] = '\0';                    // kbm for g++
    _name_width = font_text_width(_name_font, temp_str);
    temp_height = font_text_height(_name_font);
    _name_height = temp_height * (num_breaks + 1);
  }
  if(_met != UNDEFINED_TIME) {
    _met_width = font_text_width(_met_font, _met_string_ptr);
    _met_height = font_text_height(_met_font);
  }
  else {
    _met_width = 0;
    _met_height = 0;
  }
}


//   Determines a default location for the MET string.

void OperatorObject::set_default_met_location() {

  _met_x = (2 * _radius) -(_met_width / 2);
  _met_y = -5;
}

//   Determines a default location for the name string.

void OperatorObject::set_default_name_location() {

  set_text_dimensions();

  if(_is_terminator)
    _name_x = (int) ((float) _radius * 1.5) - (_name_width / 2);
  else
    _name_x = _radius - (_name_width / 2);

  _name_y = _radius + (_name_height / 2);
}

//   Convenience routine for setting both locations at once.

void OperatorObject::set_default_text_location() {

  set_default_name_location();
  set_default_met_location();
}
```

```
//   Writes a text block to the drawing canvas.  Text strings
// are broken at underscores, except when in the first and
// last character position or if line contains a '.' or '('

void OperatorObject::write_block(GC draw_context, int x, int y,
    char *instring, int block_height) {
  int i, str_len, num_lines = 1, string_width, start_index = 0,
  yinc, block_index = 0;
  char *block_text[MAXTEXTLINES], temp_string[INPUT_LINE_SIZE];
  BOOLEAN ignore_underscore;

  if(instring != NULL) {
    str_len = strlen(instring);
    if(str_len != 0) {
      ignore_underscore = (strchr(instring,'.') != 0) ||
                  (strchr(instring,'(') != 0);
      for(i = 0; i < str_len; i++)
        if ((instring[i] == '_') && !ignore_underscore) {
          if((i != 0) && (i != (str_len - 1))) {
            string_width = i - start_index + 1;
            strncpy(temp_string, &(instring[start_index]),
                    string_width);
            temp_string[string_width] = '\0';              // kbm for g++
            block_text[num_lines - 1] = dup_str(temp_string);
            start_index = i + 1;
            num_lines++;
          }
        }
      if(start_index < str_len) {
        string_width = i - start_index + 1;
        strncpy(temp_string, &(instring[start_index]),
                string_width);
        temp_string[string_width] = '\0';                  // kbm for g++
        block_text[num_lines - 1] = dup_str(temp_string);
      }
      else
        num_lines--;
      yinc = block_height / num_lines;
      for(i = 0; i < num_lines; i++) {
        str_len = strlen(block_text[i]);
        XDrawString(_display_ptr, _draw_window, draw_context, x,
                    y - (yinc * (num_lines - i - 1)),
                    block_text[i], str_len);
        XDrawString(_display_ptr, *_drawing_area_pixmap,
                    draw_context, x,
                    y - (yinc * (num_lines - i - 1)),
                    block_text[i], str_len);
        free(block_text[i]);
      }
    }
  }
```

```
}

//   Writes text on the drawing canvas, including handles if
// appropriate.  Since handles are drawn in exclusive-or mode,
// it's important to make sure that they aren't redrawn
// unless they've been erased or they need to be erased.

void OperatorObject::draw_text(DRAW_STYLE style) {
  GC draw_context;

  int xN_pos, yN_pos,
      xM_pos, yM_pos;

  xN_pos = _x + _name_x;
  yN_pos = _y + _name_y;
  xM_pos = _x + _met_x;
  yM_pos = _y + _met_y;

  if((style == SOLID) || (style == DOTTED))
    draw_context = _graphics_context;
  else
    if(style == ERASE)
      draw_context = _erase_context;
  set_font(draw_context, _name_font);

  write_block(draw_context, xN_pos, yN_pos, _name_ptr, _name_height);
  if((_name_selected) && (_name_handles_drawn == false) &&
     (style == SOLID)) {
    draw_handles(_graphics_context, xN_pos - HANDLESIZE,
                 yN_pos - _name_height - HANDLESIZE,
                 xN_pos + _name_width + HANDLESIZE,
                 yN_pos + HANDLESIZE);
    _name_handles_drawn = true;
  }
  else
    if((_name_selected) && (_name_handles_drawn == true) &&
       (style == ERASE)) {
      draw_handles(_graphics_context, xN_pos - HANDLESIZE,
                   yN_pos - _name_height - HANDLESIZE,
                   xN_pos + _name_width + HANDLESIZE,
                   yN_pos + HANDLESIZE);
      _name_handles_drawn = false;
    }
  if(_met != UNDEFINED_TIME) {
    set_font(draw_context, _met_font);
    XDrawString(_display_ptr, _draw_window, draw_context,
                xM_pos, yM_pos, _met_string_ptr,
                strlen(_met_string_ptr));
    XDrawString(_display_ptr, *_drawing_area_pixmap,
                draw_context, xM_pos, yM_pos, _met_string_ptr,
                strlen(_met_string_ptr));
    if((_met_selected) && (_met_handles_drawn == false) &&
       (style == SOLID)) {
      draw_handles(_graphics_context, xM_pos - HANDLESIZE,
                   yM_pos - _met_height - HANDLESIZE,
                   xM_pos + _met_width + HANDLESIZE,
                   yM_pos + HANDLESIZE);
      _met_handles_drawn = true;
    }
    else
      if((_met_selected) && (_met_handles_drawn == true) &&
         (style == ERASE)) {
        draw_handles(_graphics_context, xM_pos - HANDLESIZE,
                     yM_pos - _met_height - HANDLESIZE,
                     xM_pos + _met_width + HANDLESIZE,
                     yM_pos + HANDLESIZE);
        _met_handles_drawn = false;
      }

  }
}

//   Draws the operator.

void OperatorObject::draw(DRAW_STYLE style) {
  GC draw_context;

  if(_is_deleted == false) {
    if(style == SOLID)
      draw_context = _graphics_context;
    else
      if(style == DOTTED)
        draw_context = _dotted_context;
      else
        draw_context = _erase_context;

    if(_is_terminator) {
      if (_is_composite) {
if(style == SOLID) {
  XSetForeground(_display_ptr, draw_context,
 _color_table[_color]);
}
if(style != DOTTED) {
  XFillRectangle(_display_ptr, _draw_window, draw_context,
 _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
 (_radius * 3)-(COMPOSITE_SPACE*2),
 (_radius * 2)-(COMPOSITE_SPACE*2));
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,draw_context,
 _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
 (_radius * 3)-(COMPOSITE_SPACE*2),
 (_radius * 2)-(COMPOSITE_SPACE*2));
  /*
  if(style == ERASE)
    _op_handles_drawn = false;
    */
}
```

```
if(style == SOLID)
  XSetForeground(_display_ptr, draw_context, _color_table[BLACK]);

XDrawRectangle(_display_ptr, _draw_window, draw_context,
       _x, _y, _radius * 3, _radius * 2);
XDrawRectangle(_display_ptr, *_drawing_area_pixmap,
       draw_context, _x, _y, _radius * 3,
       _radius * 2);
XDrawRectangle(_display_ptr, _draw_window, draw_context,
       _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
       (_radius * 3)-(COMPOSITE_SPACE*2),
       (_radius * 2)-(COMPOSITE_SPACE*2));
XDrawRectangle(_display_ptr, *_drawing_area_pixmap, draw_context,
       _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
       (_radius * 3)-(COMPOSITE_SPACE*2),
       (_radius * 2)-(COMPOSITE_SPACE*2));
     } /* is_composite */
     else {      /* atomic */
if(style == SOLID) {
  XSetForeground(_display_ptr, draw_context,
 _color_table[_color]);
}
if(style != DOTTED) {
  XFillRectangle(_display_ptr, _draw_window, draw_context,
 _x, _y, _radius * 3, _radius * 2);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
 draw_context, _x, _y, _radius * 3,
 _radius * 2);
  /*
  if(style == ERASE)
    _op_handles_drawn = false;
    */
}
if(style == SOLID)
  XSetForeground(_display_ptr, draw_context, _color_table[BLACK]);

XDrawRectangle(_display_ptr, _draw_window, draw_context,
       _x, _y, _radius * 3, _radius * 2);
XDrawRectangle(_display_ptr, *_drawing_area_pixmap,
       draw_context, _x, _y, _radius * 3,
       _radius * 2);
     }

     if((_is_selected) && (_op_handles_drawn == false) &&
        (style == SOLID)) {
       draw_handles(_graphics_context, _x, _y,
                    _x + (3 * _radius), _y + (2 * _radius));
       _op_handles_drawn = true;
     }
     else
if((_is_selected) && (_op_handles_drawn == true) &&
   (style == ERASE)) {
  draw_handles(_graphics_context, _x, _y,
       _x + (3 * _radius), _y + (2 * _radius));
  _op_handles_drawn = false;
}
    }
    else {        /* An Operator, not a Terminator */
      if(_is_composite) {
        if(style == SOLID) {
          XSetForeground(_display_ptr, draw_context,
                         _color_table[_color]);
        }
        if(style != DOTTED) {
          XFillArc(_display_ptr, _draw_window, draw_context,
                   _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
                   (_radius - COMPOSITE_SPACE) * 2,
                   (_radius - COMPOSITE_SPACE) * 2,
                   CIRCLE_BEGIN, FULL_CIRCLE);
          XFillArc(_display_ptr, *_drawing_area_pixmap,
                   draw_context, _x + COMPOSITE_SPACE,
                   _y + COMPOSITE_SPACE,
                   (_radius - COMPOSITE_SPACE) * 2,
                   (_radius - COMPOSITE_SPACE) * 2,
                   CIRCLE_BEGIN, FULL_CIRCLE);
        }
        if(style == SOLID) {
          XSetForeground(_display_ptr, draw_context, _color_table[BLACK]);
        }
        XDrawArc(_display_ptr, _draw_window, draw_context, _x,
                 _y, _radius * 2, _radius * 2, CIRCLE_BEGIN,
                 FULL_CIRCLE);
        XDrawArc(_display_ptr, *_drawing_area_pixmap,
                 draw_context, _x, _y, _radius * 2, _radius *2,
                 CIRCLE_BEGIN, FULL_CIRCLE);
        XDrawArc(_display_ptr, _draw_window, draw_context,
                 _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
                 (_radius - COMPOSITE_SPACE) * 2,
                 (_radius - COMPOSITE_SPACE) * 2, CIRCLE_BEGIN,
                 FULL_CIRCLE);
        XDrawArc(_display_ptr, *_drawing_area_pixmap,
                 draw_context, _x + COMPOSITE_SPACE,
                 _y + COMPOSITE_SPACE,
                 (_radius - COMPOSITE_SPACE) * 2,
                 (_radius - COMPOSITE_SPACE) * 2, CIRCLE_BEGIN,
                 FULL_CIRCLE);
      }
      else {
        if(style == SOLID) {
          XSetForeground(_display_ptr, draw_context,
          _color_table[_color]);
        }
        if(style != DOTTED) {
          XFillArc(_display_ptr, _draw_window, draw_context,
                   _x, _y, _radius * 2, _radius * 2,
                   CIRCLE_BEGIN, FULL_CIRCLE);
```

```
          XFillArc(_display_ptr, *_drawing_area_pixmap,
                   draw_context, _x, _y, _radius * 2,
                   _radius * 2, CIRCLE_BEGIN, FULL_CIRCLE);
        }
        if(style == SOLID) {
          XSetForeground(_display_ptr, draw_context, _color_table[BLACK]);
        }
        XDrawArc(_display_ptr, _draw_window, draw_context,
                 _x, _y, _radius * 2, _radius * 2, CIRCLE_BEGIN,
                 FULL_CIRCLE);
        XDrawArc(_display_ptr, *_drawing_area_pixmap,
                 draw_context, _x, _y, _radius * 2, _radius *2,
                 CIRCLE_BEGIN, FULL_CIRCLE);
      }
      if((_is_selected) && (_op_handles_drawn == false) &&
         (style == SOLID)) {
        draw_handles(_graphics_context, _x, _y,
                     _x + (2 * _radius), _y + (2 * _radius));
        _op_handles_drawn = true;
      }
      else
        if((_is_selected) && (_op_handles_drawn == true) &&
           (style == ERASE)) {
          draw_handles(_graphics_context, _x, _y,
                       _x + (2 * _radius), _y + (2 * _radius));
          _op_handles_drawn = false;
        }
    }
    draw_text(style);
  }
}

void OperatorObject::erase() {

  draw(ERASE);
}

void OperatorObject::erase_text() {

  draw_text(ERASE);
}

//   Returns true if the coordinates are located within either
// the operator or one of its text strings.

BOOLEAN OperatorObject::hit(int x, int y) {

  int xN_pos, yN_pos,
      xM_pos, yM_pos;

  xN_pos = _x + _name_x;
  yN_pos = _y + _name_y;
  xM_pos = _x + _met_x;
  yM_pos = _y + _met_y;

  if(_is_deleted)
    return false;
  else {
    if(_name_ptr != NULL) {
      if(strlen(_name_ptr) != 0)
        if(((x >= xN_pos) && (x <= (xN_pos + _name_width))) &&
           (y >= yN_pos - _name_height) && (y <= yN_pos)) {
          _name_selected = true;
          return true;
        }
    }
    if(_met_string_ptr != NULL) {
      if(strlen(_met_string_ptr) != 0)
        if(((x >= xM_pos) && (x <= (xM_pos + _met_width))) &&
           (y >= yM_pos - _met_height) && (y <= yM_pos)) {
          _met_selected = true;
          return true;
        }
    }
    if(_is_terminator) {
      if(((x >= _x) && (x <= (_x + (3 * _radius)))) &&
         (y >= _y) && (y <= (_y + (2 * _radius))))
        return true;
      else
        return false;
    }
    else {
      if(((x >= _x) && (x <= (_x + (2 * _radius)))) &&
         (y >= _y) && (y <= (_y + (2 * _radius))))
        return true;
      else
        return false;
    }
  }
}

BOOLEAN OperatorObject::over(int x, int y) {

  int xN_pos, yN_pos,
      xM_pos, yM_pos;

  xN_pos = _x + _name_x;
  yN_pos = _y + _name_y;
  xM_pos = _x + _met_x;
  yM_pos = _y + _met_y;

  if(_is_deleted)
    return false;
  else {
    if(_name_ptr != NULL) {
      if(strlen(_name_ptr) != 0)
```

```
        if(((x >= xN_pos) && (x <= (xN_pos + _name_width))) &&
           (y >= yN_pos - _name_height) && (y <= yN_pos)) {
          return true;
        }
    }
    if(_met_string_ptr != NULL) {
      if(strlen(_met_string_ptr) != 0)
        if(((x >= xM_pos) && (x <= (xM_pos + _met_width))) &&
           (y >= yM_pos - _met_height) && (y <= yM_pos)) {
          return true;
        }
    }
    if(_is_terminator) {
      if(((x >= _x) && (x <= (_x + (3 * _radius)))) &&
         (y >= _y) && (y <= (_y + (2 * _radius))))
        return true;
      else
        return false;
    }
    else {
      if(((x >= _x) && (x <= (_x + (2 * _radius)))) &&
         (y >= _y) && (y <= (_y + (2 * _radius))))
        return true;
      else
        return false;
    }
  }
}

//   Returns the coordinates of the center of the given
// operator.

XYPAIR OperatorObject::center() {
  XYPAIR temp_pair;

  if(_is_terminator) {
    temp_pair.x = _x + (int) ((float) _radius * 1.5);
    temp_pair.y = _y + _radius;
  }
  else {
    temp_pair.x = _x + _radius;
    temp_pair.y = _y + _radius;
  }
  return temp_pair;
}

//   Given the last coordinate, returns the point on the
// circumference of the operator where streams should begin
// or end.

XYPAIR OperatorObject::intercept(int x, int y) {
  int distance;
  float slope;
  XYPAIR temp_pair, obj_center;

  obj_center = center();
  if(_is_terminator) {
    slope = (float) (y - obj_center.y) /
                                    (float) (x - obj_center.x);
    if(fabs(slope) >= (2.0 / 3.0)) {
      if(y <= _y)
        temp_pair.y = _y;
      else
        temp_pair.y = _y + (_radius * 2);
      temp_pair.x = (int) ((float) (temp_pair.y - obj_center.y) /
                                       slope) +  obj_center.x;
    }
    else {
      if(x <= _x)
        temp_pair.x = _x;
      else
        temp_pair.x = _x + (_radius * 3);
      temp_pair.y = (int) ((float) (temp_pair.x - obj_center.x) *
                                        slope) + obj_center.y;
    }
  }
  else {
    distance = (int) sqrt(((x - obj_center.x) * (x - obj_center.x)) +
                     (y - obj_center.y) * (y - obj_center.y));
    temp_pair.x = obj_center.x + (int)
                  (((float) _radius/ (float) distance) *
                                    (float) (x - obj_center.x));
    temp_pair.y = obj_center.y + (int)
                   (((float) _radius/(float) distance) *
                     (float) (y - obj_center.y));
  }
  return temp_pair;
}

//   Moves the operator the given distance.

void OperatorObject::move(int x, int y) {

  _x += x;
  _y += y;
  if(_x < 0)
    _x = 0;
  if(_y < 0)
    _y = 0;
}

//   Relocates the operator to the given position.  x and y
// represent the center of the operator.

void OperatorObject::set_location(int x, int y) {
```

```
  if(_is_terminator)
    _x = x - (int) ((float) _radius * 1.5);
  else
    _x = x - _radius;
  _y = y - _radius;
  if(_x < 0)
    _x = 0;
  if(_y < 0)
    _y = 0;
  reset_handles_drawn_state();
}

//   Included for symmetry.  Streams delete themselves when
// their operators are deleted, so this function is included
// for possible future use.

void OperatorObject::delete_notify(CLASS_DEF class_type,
                      OP_ID deleted_object_id) {}

void OperatorObject::name(char *new_name) {

  free(_name_ptr);
  _name_ptr = dup_str(new_name);
  set_text_dimensions();
}

void OperatorObject::unselect() {

  erase();
  _is_selected = false;
  _name_selected = false;
  _met_selected = false;
  draw(SOLID);
}

void OperatorObject::select() {

  _is_selected = true;
  draw(SOLID);
}

//   Returns true if one of the handles is in the given
// location.

BOOLEAN OperatorObject::hit_handle(int x, int y) {

  if((_x <= x) && (x <= (_x + HANDLESIZE)) &&
                      (_y <= y) && (y <= (_y + HANDLESIZE))) {
    _handle_selected = UPPERLEFT;
    return true;
  }
  if((_x <= x) && (x <= (_x + HANDLESIZE)) &&
     ((_y + 2 * _radius - HANDLESIZE)<= y) &&
      (y <= (_y + 2 * _radius))) {
    _handle_selected = LOWERLEFT;
    return true;
  }
  if(_is_terminator) {
    if(((_x + 3 * _radius - HANDLESIZE) <= x) &&
       (x <= (_x + 3 * _radius)) &&
       (_y <= y) && (y <= (_y + HANDLESIZE))) {
      _handle_selected = UPPERRIGHT;
      return true;
    }
    if(((_x + 3 * _radius - HANDLESIZE) <= x) &&
       (x <= (_x + 3 * _radius)) &&
       ((_y + 2 * _radius - HANDLESIZE) <= y) &&
        (y <= (_y + 2 * _radius))) {
      _handle_selected = LOWERRIGHT;
      return true;
    }
  }
  else {
    if(((_x + (_radius * 2) - HANDLESIZE) <= x) &&
       (x <= (_x +  2 * _radius)) &&
       (_y <= y) && (y <= (_y + HANDLESIZE))) {
      _handle_selected = UPPERRIGHT;
      return true;
    }
    if(((_x + 2 * _radius - HANDLESIZE) <= x) &&
       (x <= (_x + 2 * _radius)) &&
       ((_y + 2 * _radius - HANDLESIZE) <= y) &&
       (y <= (_y + 2 * _radius))) {
      _handle_selected = LOWERRIGHT;
      return true;
    }
  }
  _handle_selected = NONE;
  return false;
}

//   When the handle is dragged, the operator is resized.

void OperatorObject::move_handle(int x, int y) {
  int radius_change = y / 2;
//used to eliminate "floating" due to roundoff

  draw(DOTTED);
  if(_handle_selected == UPPERLEFT) {
    _radius -= radius_change;
    if(_radius >= MINRADIUS) {
      if(_is_terminator)
        _x +=  3 * radius_change;
      else
        _x +=  2 * radius_change;
      _y +=  2 * radius_change;
```

```
    }
    else
      _radius += radius_change;
  }
  else
  if(_handle_selected == UPPERRIGHT) {
    _radius -= radius_change;
    if(_radius >= MINRADIUS)
      _y +=  2 * radius_change;
    else
      _radius += radius_change;
  }
  else
    if(_handle_selected == LOWERRIGHT) {
      _radius += radius_change;
      if(_radius < MINRADIUS)
        _radius -= radius_change;
    }
    else
      if(_handle_selected == LOWERLEFT) {
        _radius += radius_change;
        if(_radius >= MINRADIUS) {
          if(_is_terminator)
            _x -=  3 * radius_change;
          else
            _x -=  2 * radius_change;
        }
        else
          _radius -= radius_change;
      }
//   Places the text in default location. May not be desired.
//  set_default_text_location();
  draw(DOTTED);
}

//   Changes the MET to that entered in the properties dialog
// box.

void OperatorObject::set_met(int met, int unit) {
  int old_met = _met;

  _met      = met;
  _met_unit = unit;
  free(_met_string_ptr);
  _met_string_ptr = time_with_units(met, unit);        // @4
  set_text_dimensions();
  if(old_met == UNDEFINED_TIME)
    set_default_met_location();
}

//   Changes the font of the selected text string.

void OperatorObject::set_object_font(int font_id) {

  if(_name_selected)
    _name_font = font_id;
  if(_met_selected)
    _met_font = font_id;
  set_text_dimensions();
}

//   Moves the text.

void OperatorObject::move_text(int x, int y) {

  erase_text();
  if (_name_selected) {
    _name_x += x;
    _name_y += y;
  }
  else
    if (_met_selected) {
      _met_x += x;
      _met_y += y;
    }
  draw_text(SOLID);
}

//   Handles undeleted operators.

void OperatorObject::undelete_notify(CLASS_DEF class_type,
                                     OP_ID deleted_obj_id) {

  if((class_type == OPERATOROBJECT) &&
      (deleted_obj_id == _id)) {
    _is_deleted      = false;
    _is_modified     = true;
    _is_selected     = false;
    _name_selected   = false;
    _met_selected    = false;
    _handle_selected = NONE;
  }
}

void OperatorObject::reset_handles_drawn_state() {

  _op_handles_drawn   = false;
  _name_handles_drawn = false;
  _met_handles_drawn  = false;
}

int OperatorObject::text_height() {

  if(_name_selected)
    return _name_height;
  else
```

```
    if(_met_selected)
      return _met_height;
    else
      return 0;
}

int OperatorObject::text_width() {

  if(_name_selected)
    return _name_width;
  else
    if(_met_selected)
      return _met_width;
    else
      return 0;
}

//   Moves the text to the desired location.

void OperatorObject::text_locate(int x, int y) {

  if(_met_selected) {
    _met_x = (x - _x) - (_met_width  / 2);
    _met_y = (y - _y) + (_met_height / 2);
  }
  else
    if(_name_selected) {
      _name_x = (x - _x) - (_name_width  / 2);
      _name_y = (y - _y) + (_name_height / 2);
    }
}

void OperatorObject::set_handle_selected(int handle) {

  _handle_selected = handle;
  _is_selected = true;
}


void OperatorObject::read_from(OPERATOR op) {

  _id           = op->id;
  _op_num               = op->op_num;                    // @5
  _x          = op->x;
  _y          = op->y;
  _radius          = op->radius;
  _color          = op->color;
  _name_ptr          = dup_str(op->label);
  _name_font          = op->label_font;
  _name_x          = op->label_x_offset;
  _name_y          = op->label_y_offset;
  _timing_type          = op->timing_type;
  _met_font          = op->met_font;
  set_met(op->met, op->met_unit);
  _met_x            = op->met_x_offset;
  _met_y            = op->met_y_offset;
  _met_reqmts          = id_list_copy(op->met_reqmts);
  _period          = op->period;
  _period_unit          = op->period_unit;
  _period_reqmts          = id_list_copy(op->period_reqmts);
  _fw          = op->fw;
  _fw_unit          = op->fw_unit;
  _fw_reqmts          = id_list_copy(op->fw_reqmts);
  _mrt          = op->mrt;
  _mrt_unit          = op->mrt_unit;
  _mrt_reqmts          = id_list_copy(op->mrt_reqmts);
  _mcp          = op->mcp;
  _mcp_unit          = op->mcp_unit;
  _mcp_reqmts          = id_list_copy(op->mcp_reqmts);
  _trigger_type          = op->trigger_type;
  _trigger_set           = id_list_copy(op->trigger_set);        // @5
  _trigger_if_condition  = dup_str(op->if_condition);            // @2
  _trigger_reqmts        = id_list_copy(op->trigger_reqmts);
  _output_guard_list     = dup_str(op->output_guard_list);       // @2
  _exception_list        = dup_str(op->exception_list);          // @2
  _timer_op_list         = dup_str(op->timer_op_list);           // @2
  _key_word_list         = id_list_copy(op->key_word_list);      // @2
  _operator_informal_desc = dup_str(op->operator_informal_desc); // @2
  _operator_formal_desc  = dup_str(op->operator_formal_desc);    // @2
  _operator_impl_lang    = dup_str(op->operator_impl_lang);
  _is_composite          = op->is_composite;
  is_terminator(op->is_terminator);    // correct met if wrong
  _is_new                = false;     // SDE status is overwritten
  _is_modified           = false;
  _is_deleted            = false;

  _is_selected           = false;
  _handle_selected       = NONE;
  _name_selected         = false;
  _met_selected          = false;

  set_text_dimensions();
  reset_handles_drawn_state();
}

void OperatorObject::write_to(OPERATOR on) {

  on->id = _id;
  on->op_num           = _op_num;             // @5
  on->x = _x;
  on->y = _y;
  on->radius = _radius;
  on->color = _color;

  on->label            = dup_str(_name_ptr);       // kbm
  on->label_font = _name_font;
```

```
  on->label_x_offset = _name_x;
  on->label_y_offset = _name_y;

  on->timing_type = _timing_type;
  on->met = _met;
  on->met_unit = _met_unit;
  on->met_font = _met_font;
  on->met_x_offset = _met_x;
  on->met_y_offset = _met_y;
  on->met_reqmts = id_list_copy(_met_reqmts);
  on->period = _period;
  on->period_unit = _period_unit;
  on->period_reqmts = id_list_copy(_period_reqmts);
  on->fw = _fw;
  on->fw_unit = _fw_unit;
  on->fw_reqmts = id_list_copy(_fw_reqmts);
  on->mrt = _mrt;
  on->mrt_unit = _mrt_unit;
  on->mrt_reqmts = id_list_copy(_mrt_reqmts);
  on->mcp = _mcp;
  on->mcp_unit = _mcp_unit;
  on->mcp_reqmts = id_list_copy(_mcp_reqmts);

  on->trigger_type = _trigger_type;
  on->trigger_set = id_list_copy(_trigger_set);
  on->if_condition  = dup_str(_trigger_if_condition);
  on->trigger_reqmts = id_list_copy(_trigger_reqmts);

  on->output_guard_list = dup_str(_output_guard_list);
  on->exception_list  = dup_str(_exception_list);
  on->timer_op_list  = dup_str(_timer_op_list);
  on->key_word_list = id_list_copy(_key_word_list);
  on->operator_informal_desc  = dup_str(_operator_informal_desc);
  on->operator_formal_desc    = dup_str(_operator_formal_desc);

  on->operator_impl_lang = dup_str(_operator_impl_lang);

  on->is_composite = _is_composite;
  on->is_terminator = _is_terminator;
  on->is_new = _is_new;
  on->is_modified = _is_modified;
  on->is_deleted = _is_deleted;
}


void OperatorObject::release() {

#ifdef GE_DEBUG
//  printf("OperatorObject::release called for: %s\n", _name_ptr);
#endif

  //Added by Doug Lange 9/9/96; kbm 10/24/96

  free(_name_ptr);
  _name_ptr                = NULL;

  free(_met_string_ptr);
  _met_string_ptr          = NULL;

  free(_trigger_if_condition);
  _trigger_if_condition    = NULL;

  free(_output_guard_list);
  _output_guard_list       = NULL;

  free(_exception_list);
  _exception_list          = NULL;

  free(_timer_op_list);
  _timer_op_list           = NULL;

  free(_operator_informal_desc);
  _operator_informal_desc  = NULL;

  free(_operator_formal_desc);
  _operator_formal_desc    = NULL;

  free(_operator_impl_lang);
  _operator_impl_lang      = NULL;

  id_list_release(_met_reqmts);        _met_reqmts     = NULL;
  id_list_release(_period_reqmts);     _period_reqmts  = NULL;
  id_list_release(_fw_reqmts);         _fw_reqmts      = NULL;
  id_list_release(_mrt_reqmts);        _mrt_reqmts     = NULL;
  id_list_release(_mcp_reqmts);        _mcp_reqmts     = NULL;
  id_list_release(_trigger_set);       _trigger_set    = NULL;
  id_list_release(_trigger_reqmts);    _trigger_reqmts = NULL;
  id_list_release(_key_word_list);     _key_word_list  = NULL;

}


void OperatorObject::copy(OperatorObject *src) {

  _id           = src->_id;
  _op_num               = src->_op_num;
  _x           = src->_x;
  _y           = src->_y;
  _radius           = src->_radius;
  _color          = src->_color;
  _name_ptr          = dup_str(src->_name_ptr);
  _name_font           = src->_name_font;
  _name_x           = src->_name_x;
  _name_y           = src->_name_y;
  _timing_type          = src->_timing_type;
  _met_font           = src->_met_font;
```

```
  _met                    = src->_met;
  _met_unit               = src->_met_unit;
  _met_string_ptr         = dup_str(src->_met_string_ptr);
  _met_x            = src->_met_x;
  _met_y            = src->_met_y;
  _met_reqmts           = id_list_copy(src->_met_reqmts);
  _period           = src->_period;
  _period_unit          = src->_period_unit;
  _period_reqmts          = id_list_copy(src->_period_reqmts);
  _fw           = src->_fw;
  _fw_unit            = src->_fw_unit;
  _fw_reqmts          = id_list_copy(src->_fw_reqmts);
  _mrt          = src->_mrt;
  _mrt_unit           = src->_mrt_unit;
  _mrt_reqmts           = id_list_copy(src->_mrt_reqmts);
  _mcp          = src->_mcp;
  _mcp_unit         = src->_mcp_unit;
  _mcp_reqmts           = id_list_copy(src->_mcp_reqmts);
  _trigger_type           = src->_trigger_type;
  _trigger_set            = id_list_copy(src->_trigger_set);
  _trigger_if_condition   = dup_str(src->_trigger_if_condition);
  _trigger_reqmts         = id_list_copy(src->_trigger_reqmts);
  _output_guard_list      = dup_str(src->_output_guard_list);
  _exception_list         = dup_str(src->_exception_list);
  _timer_op_list          = dup_str(src->_timer_op_list);
  _key_word_list          = id_list_copy(src->_key_word_list);
  _operator_informal_desc = dup_str(src->_operator_informal_desc);
  _operator_formal_desc   = dup_str(src->_operator_formal_desc);
  _operator_impl_lang     = dup_str(src->_operator_impl_lang);
  _is_composite           = src->_is_composite;
  _is_terminator          = src->_is_terminator;
  _is_new                 = src->_is_new;
  _is_modified            = src->_is_modified;
  _is_deleted             = src->_is_deleted;

  _is_selected            = src->_is_selected;
  _handle_selected        = src->_handle_selected;
  _name_selected          = src->_name_selected;
  _met_selected           = src->_met_selected;

  _name_width             = src->_name_width;
  _name_height            = src->_name_height;
  _met_width              = src->_met_width;
  _met_height             = src->_met_height;

  _op_handles_drawn       = src->_op_handles_drawn;
  _name_handles_drawn     = src->_name_handles_drawn;
  _met_handles_drawn      = src->_met_handles_drawn;
}


void OperatorObject::initialize() {
```

```
_id           = UNDEFINED_OPNUM;
_op_num                 = UNDEFINED_OPNUM;
_x          = 0;
_y          = 0;
_radius           = 0;
_color          = 0;
_name_ptr           = dup_str("");
_name_font            = GraphObject::_default_font;
_name_x           = 0;
_name_y           = 0;
_timing_type            = NTC;
_met          = UNDEFINED_TIME;
_met_unit           = MS;
_met_string_ptr         = dup_str("");
_met_font           = GraphObject::_default_font;
_met_x            = 0;
_met_y            = 0;
_met_reqmts           = NULL;
_period           = UNDEFINED_TIME;
_period_unit            = MS;
_period_reqmts          = NULL;
_fw           = UNDEFINED_TIME;
_fw_unit            = MS;
_fw_reqmts            = NULL;
_mrt          = UNDEFINED_TIME;
_mrt_unit           = MS;
_mrt_reqmts           = NULL;
_mcp          = UNDEFINED_TIME;
_mcp_unit       , = MS;
_mcp_reqmts           = NULL;
_trigger_type           = UNPROTECTED;
_trigger_set            = NULL;
_trigger_if_condition   = dup_str("");
_trigger_reqmts         = NULL;
_output_guard_list      = dup_str("");
_exception_list         = dup_str("");
_timer_op_list          = dup_str("");
_key_word_list          = NULL;
_operator_informal_desc = dup_str("");
_operator_formal_desc   = dup_str("");
_operator_impl_lang     = dup_str(DEFAULT_IMPL_LANG);
_is_composite           = false;
is_terminator(false);
_is_new                 = false;
_is_modified            = false;
_is_deleted             = false;

_is_selected            = false;
_handle_selected        = NONE;
_name_selected          = false;
_met_selected           = false;

set_text_dimensions();
```

```
  set_default_text_location();
  set_default_met_location();
  reset_handles_drawn_state();

}


void OperatorObject::is_terminator(BOOLEAN is_terminator) {

  _is_terminator = is_terminator;

  if (_is_terminator) {
    set_met(0, MS);
  }
  else {
    /* Do not change if not zero */
    if (_met == 0) {
      set_met(UNDEFINED_TIME, MS);
    }
  }
}
```

```
#include <Xm/Xm.h>
#include "operator_object.h"

#ifndef OPERATOR_PROPERTY_MENU_H
#define OPERATOR_PROPERTY_MENU_H

void operator_property_dialog(Widget parent_widget,
      OperatorObject *op_being_updated,
      int x, int y, BOOLEAN parent_terminator,
      ID_LIST avail_impl_langs,
      GraphObjectList *graphic_list);

#endif /* OPERATOR_PROPERTY_MENU_H */
```

```
/*****************************************************************
 *
 * Project:      PSDL Editor
 * Assembly:     GUI (Graphics User Interface)
 * SubAssem:     Operator Properties
 *
 * Programmer:   Ken Moeller
 * Langauge:     C++
 *
 * Description:  This package is responsible for producing the pop-up
 *      window for displaying and maintaining the properties of an operator.
 *      operator_property_dialog is the main routine.  Several callback
 *      routines are also provided.
 *
 *      The OperatorObject that is being updated is provided by
 *      op_being_updated, which is a global symbol.  It is expected to
 *      be set prior to calling operator_property_dialog.
 *
 *
 *
 * Modules:
 *
 * operator_name_cb
 *      met_cb
 *      met_unit_cb
 *      period_cb
 *      period_unit_cb
 *      fw_cb
 *      fw_unit_cb
 *      mcp_cb
 *      mcp_unit_cb
 *      mrt_cb
 *      mrt_unit_cb
 *      operator_ok_cb
 *      operator_cancel_cb
 *      operator_help_cb
 *      operator_property_dialog
 *
 *
 * Globals/Static:
 *
 *      op_being_updated
 *
 *      temp_op_info
 *
 *      op_dialog
 *
 * History:
 *
 * Id   Date      Author        Change
 * @1   yy/mm/dd  Ken Moeller
 *
 *
 * Suggested Modifications:
 *
 *   Get rid of op_being_updated from graph_editor and pass it to
 *   operator_property_dialog as an argument.
 *
 *****************************************************************/

#include <stdlib.h>
#include <malloc.h>
#include <memory.h>

#include <X11/Xatom.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/Form.h>
#include <Xm/LabelG.h>
#include <Xm/List.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectioB.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/ToggleBG.h>
#include <Xm/Xm.h>

#include "ge_defs.h"
#include "ge_interface.h"
#include "ge_utilities.h"
#include "gettopshell.h"
#include "graph_editor.h"
#include "graph_object_list.h"
#include "operator_object.h"
#include "operator_property_menu.h"
#include "warning.h"
#include "windows.h"
#include "action_area.h"
#include "build_option.h"

// Popup Window Parameters
#define WIN_WIDTH   390
#define WIN_HEIGHT  645
#define TAB1        5
#define TAB2        20
#define TAB3        140
#define TAB4        150
#define TAB5        220
#define TAB6        283
#define TAB_LABEL   35
```

```
#define TAB_VALUE   90
#define TAB_UNIT    170
#define TAB_REQMT   270
#define ROW1        5
#define ROW2        45
#define ROW3        85
#define ROW4        125
#define ROW5        155
#define ROW6        185
#define ROW7        225
#define ROW8        260
#define ROW9        295
#define ROW10       330
#define ROW11       370
#define ROW12       400
#define ROW13       430
#define ROW14       470
#define ROW15       510
#define ROW16       550

#define MET         1
#define PERIOD      2
#define FW          3
#define MCP         4
#define MRT         5
#define TRIGGER     6

// Operator Type
#define OPERATOR_TYPE   0
#define TERMINATOR_TYPE 1

static OperatorObject *temp_op_info; // Access required in callback routines
static Widget op_dialog;             // Access required in callback routines

static Widget oper_label;     // Operator or Terminal   - selectable
static Widget oper_name;
static Widget impl_lang;      // eg Ada, TAE, PSDL      - selectable
static Widget trigger;
static Widget trigger_list = NULL;
Widget trigger_if_cond, trigger_if_value = NULL;
Widget trigger_req_by;
Widget timing;           // NTC, Periodic, Sporadic
Widget met_label,    met,    met_unit,    met_unit_label,    met_req_by;
Widget period_label, period, period_unit, period_unit_label, period_req_by;
Widget mrt_label,    mrt,    mrt_unit,    mrt_unit_label,    mrt_req_by;
Widget mcp_label,    mcp,    mcp_unit,    mcp_unit_label,    mcp_req_by;
Widget fw_label,     fw,     fw_unit,     fw_unit_label,     fw_req_by;
Widget fw_label2;
Widget output_guard, output_guard_value = NULL;
Widget exception,    exception_value    = NULL;
Widget timer_op,     timer_op_value     = NULL;
Widget spec_keyword_button, spec_informal_button, spec_formal_button;
Widget ok_button, cancel_button, help_button;

GraphObjectList *opDialogGraphicList;

char* if_condition_prefix = NULL;
char* output_guard_prefix = NULL;
char* exception_prefix    = NULL;
char* timer_op_prefix     = NULL;

BOOLEAN op_name_error, op_met_error, op_period_error,
        op_mrt_error,      op_mcp_error, op_fw_error;


ID_LIST Global_avail_impl_langs;

typedef struct widget_data {
  Widget    widget;
  ID_LIST*  id_list_adr;
  void*     display_cb;
  Widget    disp_widget;
} Widget_Data_Node, *Widget_Data_Ptr;


typedef void (*display_fun)(Widget, Widget_Data_Ptr);

void id_list_dialog(Widget parent, XtPointer id_list_ptr,
    XtPointer display_cb, Widget disp_widget);
static void display_timing(int timing_type);


void req_by_label(Widget disp_w, ID_LIST id_list_ptr) {

  XmString tmp;

  if (id_list_ptr != NULL)
    tmp = XmStringCreateSimple("Required By ...   ");
  else
    tmp = XmStringCreateSimple("Required By       ");

  XtVaSetValues(disp_w, XmNlabelString, tmp , NULL);

  XmStringFree(tmp);

}

void trigger_label(Widget disp_w, ID_LIST id_list_ptr) {

  char *stream_list_ptr = NULL;
  XmString tmp;
  char buffer[27] = "                          ";

  if (id_list_ptr != NULL) {
    id_list_str_eps(id_list_ptr, &stream_list_ptr,25);
    memcpy(buffer, stream_list_ptr, strlen(stream_list_ptr));
```

```
    tmp = XmStringCreateSimple(buffer);
    free(stream_list_ptr);      stream_list_ptr = NULL;
    XtVaSetValues(disp_w, XmNalignment, XmALIGNMENT_BEGINNING, NULL);
  }
  else {
    tmp = XmStringCreateSimple("Stream List");
    XtVaSetValues(disp_w, XmNalignment, XmALIGNMENT_CENTER, NULL);
  }

  XtVaSetValues(disp_w, XmNlabelString, tmp , NULL);
  XtVaSetValues(disp_w, XmNwidth, 170, NULL);

  XmStringFree(tmp);

}

void keyword_label(Widget disp_w, ID_LIST id_list_ptr) {

  XmString tmp;

  if (id_list_ptr != NULL)
    tmp = XmStringCreateSimple("  Keywords ... ");
  else
    tmp = XmStringCreateSimple("   Keywords    ");

  XtVaSetValues(disp_w, XmNlabelString, tmp , NULL);

  XmStringFree(tmp);

}


void informal_label(Widget disp_w, char** char_ptr_adr) {

  XmString tmp;

  if ((*char_ptr_adr != NULL) && (**char_ptr_adr != '\0'))
    tmp = XmStringCreateSimple("Informal Desc...");
  else
    tmp = XmStringCreateSimple(" Informal Desc  ");

  XtVaSetValues(disp_w, XmNlabelString, tmp , NULL);

  XmStringFree(tmp);

}

void formal_label(Widget disp_w, char** char_ptr_adr) {

  XmString tmp;

  if ((*char_ptr_adr != NULL) && (**char_ptr_adr != '\0'))
    tmp = XmStringCreateSimple(" Formal Desc... ");
  else
    tmp = XmStringCreateSimple("  Formal Desc   ");

  XtVaSetValues(disp_w, XmNlabelString, tmp , NULL);

  XmStringFree(tmp);

}

static void trigger_display_cb(Widget w,
       Widget_Data_Ptr client_data) {

  ID_LIST*  id_list_adr = client_data->id_list_adr;
  Widget    disp_w      = client_data->disp_widget;

  trigger_label(disp_w, *id_list_adr);
}


static void req_by_display_cb(Widget w,
      Widget_Data_Ptr client_data) {

  ID_LIST*  id_list_adr = client_data->id_list_adr;
  Widget    disp_w      = client_data->disp_widget;

  req_by_label(disp_w, *id_list_adr);
}


static void keyword_display_cb(Widget w,
       Widget_Data_Ptr client_data) {

  ID_LIST*  id_list_adr = client_data->id_list_adr;
  Widget    disp_w      = client_data->disp_widget;

  keyword_label(disp_w, *id_list_adr);
}


static void req_by_cb(Widget w, XtPointer client_data,
                      XtPointer cb_struct_ptr) {

  id_list_dialog(w, client_data, &req_by_display_cb, w);

  return;
}


static void keyword_cb(Widget w, XtPointer client_data,
XtPointer call_data) {

  id_list_dialog(w, client_data, &keyword_display_cb, w);
```

```
}


static void op_prop_informal_desc_ok_cb(Widget w, XtPointer client_data,
  XtPointer call_data) {
  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);
  temp_op_info->operator_informal_desc(text);
  free(text);

  XtDestroyWidget(GetTopShell(w));

  informal_label(spec_informal_button,
 temp_op_info->operator_informal_desc_adr());
  clear_status();
  return;
}

static void informal_desc_cb(Widget w, XtPointer client_data,
     XtPointer call_data) {

  static ActionAreaItem   action_items[] = {
    {"OK",      op_prop_informal_desc_ok_cb, NULL            },
    {"Cancel",  close_dialog,                NULL            },
    {"Help",    help_cb,                     "op_prop_informal_desc.hlp"  }
  };

  Widget   dialog, pane, rc, text_w, action_a;
  XmString string;

  char *text;

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 w,
 XmNtitle, "Informal Design Description",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog; //Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                        pane, NULL);
  string = XmStringCreateSimple("Enter or Edit Informal Description");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  text = temp_op_info->operator_informal_desc();

  int n = 0;
  Arg args[10];
  XtSetArg(args[n], XmNrows,                  12); n++;
  XtSetArg(args[n], XmNcolumns,               70); n++;
  XtSetArg(args[n], XmNscrollVertical,        true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,      false); n++;
  XtSetArg(args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,              true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,              true); n++;
  XtSetArg(args[n], XmNvalue,                 text); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);

  free(text);

  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items,
                              XtNumber(action_items));

  XtManageChild(pane);
  XtPopup(dialog, XtGrabNone);
}

static void op_prop_formal_desc_ok_cb(Widget w, XtPointer client_data,
  XtPointer call_data) {
  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);
  temp_op_info->operator_formal_desc(text);
  free(text);

  XtDestroyWidget(GetTopShell(w));

  formal_label(spec_formal_button,
       temp_op_info->operator_formal_desc_adr());

  clear_status();

  return;
}
```

```
static void formal_desc_cb(Widget w, XtPointer client_data,
   XtPointer call_data) {
  static ActionAreaItem   action_items[] = {
    {"OK",      op_prop_formal_desc_ok_cb, NULL            },
    {"Cancel",  close_dialog,              NULL            },
    {"Help",    help_cb,                   "op_prop_formal_desc.hlp"  }
  };

  Widget   dialog, pane, rc, text_w, action_a;
  XmString string;

  char *text;

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
      w, XmNtitle, "Formal Design Description",
      XmNdeleteResponse, XmDESTROY,
      NULL);

  action_items[1].data = (XtPointer)dialog;//Set cancel buttons client_data

   pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
      XmNsashWidth,   1,
      XmNsashHeight,  1,
      NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                        pane, NULL);
  string = XmStringCreateSimple("Enter or Edit Formal Description");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
      XmNlabelString, string,
      NULL);
  XmStringFree(string);

  text = temp_op_info->operator_formal_desc();

  int n = 0;
  Arg args[10];
  XtSetArg(args[n], XmNrows,                12); n++;
  XtSetArg(args[n], XmNcolumns,             70); n++;
  XtSetArg(args[n], XmNscrollVertical,      true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,    false); n++;
  XtSetArg(args[n], XmNeditMode,            XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,            true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,            true); n++;
  XtSetArg(args[n], XmNvalue,               text); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);

  free(text);

  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  XtManageChild(pane);
  XtPopup(dialog, XtGrabNone);
}


static void trigger_list_cb(Widget w, XtPointer client_data,
                      XtPointer cb_struct_ptr) {

  id_list_dialog(w, client_data, &trigger_display_cb, w);

  return;
}


static void id_list_ok_cb(Widget w, XtPointer client_data,
        XtPointer call_data) {
  Widget list_w      = ((Widget_Data_Ptr) client_data)->widget;
  ID_LIST* id_list_adr = ((Widget_Data_Ptr) client_data)->id_list_adr;
  void* display_cb  = ((Widget_Data_Ptr) client_data)->display_cb;
  Widget disp_widget = ((Widget_Data_Ptr) client_data)->disp_widget;
  XmAnyCallbackStruct *cbs    = (XmAnyCallbackStruct *)call_data;

  display_fun  df = (display_fun) display_cb;

  int                u_bound;
  XmString           *strlist;
  char               *text;
  ID_LIST            idp, idp_head;

  XtVaGetValues(list_w,
        XmNitemCount,   &u_bound,
        XmNitems,       &strlist,
        NULL);

  if (u_bound > 0) {
    idp_head = (ID_LIST) malloc(sizeof(ID_NODE));
    idp = idp_head;
    idp->next = NULL;

    if (XmStringGetLtoR(strlist[0], XmSTRING_DEFAULT_CHARSET, &text)) {
      idp->id = text;
    }

    for (int i = 1; i < u_bound; i++) {
       idp->next = (ID_LIST) malloc(sizeof(ID_NODE));
```

```
      idp = idp->next;
      idp->next = NULL;
      if (XmStringGetLtoR(strlist[i], XmSTRING_DEFAULT_CHARSET, &text))
        idp->id = text;
    }
  } else {
    idp_head = NULL;
  }

  id_list_replace(id_list_adr, idp_head);

  (*df)(disp_widget, (Widget_Data_Ptr) client_data);

  XtDestroyWidget(GetTopShell(w));

  free((char*) client_data);

  clear_status();

  return;
}


static void id_list_cancel_cb(Widget w, XtPointer client_data,
         XtPointer call_data) {

  XtDestroyWidget(GetTopShell(w));

  free((char*) client_data);
  clear_status();

  return;
}


static void read_id_cb(Widget widget, XtPointer client_data,
    XtPointer call_data) {
  Widget list_w = (Widget)client_data;
  XmSelectionBoxCallbackStruct *cbs =
                      (XmSelectionBoxCallbackStruct *)call_data;
  char *text, *newtext;
  int u_bound;
  XmString *strlist;

  //check to make sure entry is not null
  if (XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET,
                      &newtext)) {
    if (newtext && *newtext){

      if (!valid_id(newtext)) {
warning(widget, "Invalid ID");
update_status("Illegal ID: id ::= letter {alpha_numeric}",
      RING_BELL);
return;
      }
      if (is_keyword(newtext, false)) {
warning(widget, "ID is a keyword");
update_status("ID is a keyword, change or cancel", RING_BELL);
return;
      }

      XmListAddItemUnselected(list_w, cbs->value, 0);
    }
  }
  clear_status();

  XtDestroyWidget(widget);

  return;
}


static void edit_id_cb(Widget widget, XtPointer client_data,
    XtPointer call_data) {
  Widget list_w = (Widget)client_data;
  XmSelectionBoxCallbackStruct *cbs =
                          (XmSelectionBoxCallbackStruct *)call_data;
  char *text, *newtext;
  int u_bound;
  XmString *strlist,new_items;
  int     i,*pos_list, pos_cnt;

  //check to make sure entry is not null
  if (XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET,
                      &newtext)) {
    if (newtext && *newtext){


      if (!valid_id(newtext)) {
warning(widget, "Invalid ID");
update_status("Illegal ID: id ::= letter {alpha_numeric}",
      RING_BELL);
return;
      }
      if (is_keyword(newtext, false)) {
warning(widget, "ID is a keyword");
update_status("ID is a keyword, change or cancel", RING_BELL);
return;
      }

      XmListGetSelectedPos(list_w, &pos_list, &pos_cnt);
      new_items = XmStringCreateSimple(newtext);
      for (i = 0; i < pos_cnt; i++) {
XmListReplaceItemsPos(list_w, &new_items, 1, pos_list[i]);
      }
```

```
    }
  }

  clear_status();
  XtDestroyWidget(widget);
  return;
}


static void id_list_add_cb(Widget w, XtPointer client_data,
         XtPointer call_data) {
  Widget      dialog;
  Widget      list_w = (Widget)client_data;

  XmString    t = XmStringCreateSimple("Enter New ID");
  Arg         args[5];
  int         n = 0;

  XtSetArg(args[n], XmNselectionLabelString,  t);     n++;
  XtSetArg(args[n], XmNautoUnmanage,          false); n++;
  dialog = XmCreatePromptDialog(w, "prompt", args, n);
  XmStringFree(t);

  XtAddCallback(dialog, XmNokCallback, read_id_cb, list_w);
  XtAddCallback(dialog, XmNcancelCallback, dlg_callback, NULL);
  XtSetSensitive(
    XmSelectionBoxGetChild(dialog, XmDIALOG_HELP_BUTTON), false);
  XtManageChild(dialog);

  XtPopup(XtParent(dialog), XtGrabNone);

  return;
}


static void id_list_del_cb(Widget w, XtPointer client_data,
         XtPointer call_data) {
  Widget      dialog;
  Widget      list_w = (Widget)client_data;
  int         *pos_list, pos_cnt;

  if (!XmListGetSelectedPos(list_w, &pos_list, &pos_cnt)) {
    warning(w, "Nothing Selected");
    return;
  }

  XmListDeletePos(list_w, pos_list[0]);

  return;
}


static void id_list_edit_cb(Widget w, XtPointer client_data,
       XtPointer call_data) {
  Widget      dialog;
  Widget      list_w = (Widget)client_data;

  XmString    t = XmStringCreateSimple("Enter Revised ID");
  Arg         args[5];
  int         n = 0, *pos_list, pos_cnt;

  if (!XmListGetSelectedPos(list_w, &pos_list, &pos_cnt)) {
    warning(w, "Nothing Selected");
    return;
  }

  XtSetArg(args[n], XmNselectionLabelString,  t);     n++;
  XtSetArg(args[n], XmNautoUnmanage,          false); n++;
  dialog = XmCreatePromptDialog(w, "prompt", args, n);
  XmStringFree(t);
  XtFree((char*)pos_list);

  XtAddCallback(dialog, XmNokCallback, edit_id_cb, list_w);
  XtAddCallback(dialog, XmNcancelCallback, dlg_callback, NULL);

  XtSetSensitive(
       XmSelectionBoxGetChild(dialog, XmDIALOG_HELP_BUTTON), false);
  XtManageChild(dialog);

  return;
}


void id_list_dialog(Widget parent, XtPointer id_list_ptr,
    XtPointer display_cb, Widget disp_widget) {

  static ActionAreaItem action_items[] = {
    {"OK",      id_list_ok_cb,       NULL},
    {"Cancel",  id_list_cancel_cb,   NULL},
    {"Add",     id_list_add_cb,      NULL},
    {"Delete",  id_list_del_cb,      NULL},
    {"Edit",    id_list_edit_cb,     NULL},
    {"Help",    help_cb,            "id_list.hlp"}
  };

  Widget      dialog, rc, pane, list, action_a;
  int         count = 0, i, n=0;
  ID_LIST     idp;
  Arg         args[5];
  XmString    *str,string;

  //Build list for list widget
  idp = (ID_LIST) *((ID_LIST*) id_list_ptr);

  while(idp) { // count number of IDs in list
    count++;
```

```
  idp = idp->next;
}

idp = (ID_LIST) *((ID_LIST*) id_list_ptr);

str = (XmString *) XtMalloc (count * sizeof (XmString));
for (i = 0; i < count; i++) {
  str[i] = XmStringCreateSimple(idp->id);
  idp = idp->next;
}

dialog = XtVaCreatePopupShell("dialog", xmDialogShellWidgetClass,
           XtParent(parent),
           XmNtitle, "ID List",
           XmNdeleteResponse, XmDESTROY,
           NULL);

pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
           XmNsashWidth,   1,
           XmNsashHeight,  1,
           NULL);

rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                      pane, NULL);
string = XmStringCreateSimple("Enter or Edit IDs");
XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
           XmNlabelString, string,
           NULL);
XmStringFree(string);

list = XmCreateScrolledList(rc, "ID_List", NULL, 0);
XtVaSetValues(list,
      XmNvisibleItemCount,    10,
      XmNitemCount,           count,
      XmNitems,               str,
      NULL);
XtManageChild(list);

for (i = 0; i < count; i++) {
  XmStringFree(str[i]);
}

// Make a client_data structure
Widget_Data_Ptr wd = (Widget_Data_Ptr) malloc(sizeof(Widget_Data_Node));
wd->widget      = list;
wd->id_list_adr = (ID_LIST*) id_list_ptr;
wd->display_cb  = (void *)   display_cb;
wd->disp_widget = (Widget)   disp_widget;

XtManageChild(rc);

//Set client data for "OK", "Cancel", "Add", "Del", and "Edit" buttons
action_items[0].data = (XtPointer)wd;       // OK
action_items[1].data = (XtPointer)wd;       // Cancel
action_items[2].data = (XtPointer)list;     // Add
action_items[3].data = (XtPointer)list;     // Del
action_items[4].data = (XtPointer)list;     // Edit

action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

XtManageChild(pane);
XtPopup(dialog, XtGrabNone);

return;
}


static void operator_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if (state->set) {
      temp_op_info->is_terminator( (int) which );
      display_timing(temp_op_info->timing_type());
  }

  return;
}


static void operator_name_cb(Widget w, XtPointer client_data,
     XtPointer call_data) {

  Widget temp_w =  (Widget) client_data;
  char *text;
  char *old_name;

  text = XmTextFieldGetString(temp_w);

  if (!valid_op_id(text)) {
    warning(w, "Invalid operator name (syntax or keyword)");
    update_status("Illegal operator name, correct or cancel: "
       "op_id ::= [id '.'] op_name ['(' [id_list] '|' [id_list] ')'",
  RING_BELL);
    XtFree(text);
    op_name_error = true;
    return;
  } else if ((strchr(text,'.') != NULL) && temp_op_info->is_composite()) {
    warning(w, "A Composite Operator can not be a Type");
    update_status("Composite Operator can not be a Type: "
                  "rename operator or make Automic",
      RING_BELL);
    XtFree(text);
    op_name_error = true;
    return;
  } else if ((strchr(text,'.') == NULL) &&
```

```
     !opDialogGraphicList->unique_op_id(text,temp_op_info->id())) {
    warning(w, "Simple Operator Names must be unique to level");
    update_status("Operators that are not types must have a unique name",
      RING_BELL);
    XtFree(text);
    op_name_error = true;
    return;
  }

  old_name = temp_op_info->name();

  if (text == NULL)
    temp_op_info->name("");
  else
    temp_op_info->name(text);

  if (!old_name || *old_name == '\0')
    temp_op_info->set_default_name_location();

  free(old_name);

  op_name_error = false;
  XtFree(text);

  return;
}


static void impl_lang_cb(Widget w, XtPointer client_data,
 XtPointer call_data) {

  int which =  (int) client_data;
  ID_LIST langPtr;
  int i;

  langPtr = Global_avail_impl_langs;

  for (i = 0; i < which; i++)
    langPtr = langPtr->next;

  temp_op_info->operator_impl_lang(langPtr->id);

  return;
}

static void trigger_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if (state->set)
    temp_op_info->trigger_type( (int) which );

  if (temp_op_info->trigger_type() != UNPROTECTED) {
    if (!trigger_list) {
      trigger_list = XtVaCreateManagedWidget(dup_str("Stream List"),
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB5,
XmNy, ROW4+3,
XmNwidth, 170,
NULL);
      XtAddCallback(trigger_list, XmNactivateCallback,
    trigger_list_cb,
    (XtPointer) temp_op_info->trigger_set_adr());

      trigger_label(trigger_list, *(temp_op_info->trigger_set_adr()));
    }
  }
  else {
    if (trigger_list) {
      XtDestroyWidget(trigger_list);
      trigger_list = NULL;
    }
  }
  return;
}


static void met_unit_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if (state->set)
    temp_op_info->met_unit( (int) which );

  return;
}

static void period_unit_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if (state->set)
    temp_op_info->period_unit( (int) which );

  return;
}

static void fw_unit_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if(state->set)
      temp_op_info->fw_unit( (int) which );

  return;
}
```

```
static void mrt_unit_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if(state->set)
      temp_op_info->mrt_unit( (int) which );

  return;
}


static void mcp_unit_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if(state->set)
      temp_op_info->mcp_unit( (int) which );

  return;
}


static BOOLEAN read_time_value(Widget w, int &time) {

  char    buffer[INPUT_LINE_SIZE];
  char*   text = NULL;
  int     time_value;
  BOOLEAN error_flag = false;

  text = XmTextFieldGetString(w);

  if (!valid_integer_literal(text, &time_value)) {
    time_value = UNDEFINED_TIME;
    if (white_space(text))
      error_flag = false;
    else {
      error_flag = true;
    }
  }
  else
    error_flag = false;

  // Display new value from latency
  if (time_value != UNDEFINED_TIME) {
    sprintf(buffer, "%d", time_value);
    XtVaSetValues(w, XmNvalue, buffer, NULL);
  }

  XtFree(text);

  time = time_value;
  return error_flag;
}


static void read_timing_cb(Widget w, XtPointer client_data,
   XtPointer call_data) {

  int time_value;
  BOOLEAN reset_met_location;

  reset_met_location = (temp_op_info->met() == UNDEFINED_TIME);

  switch (temp_op_info->timing_type()) {
  case NTC:

    if (!temp_op_info->is_terminator())
      temp_op_info->set_met(UNDEFINED_TIME,MS);   // Clear string

    break;

  case PERIODIC:

    if (!temp_op_info->is_terminator()) {
      op_met_error = read_time_value(met, time_value);
      temp_op_info->set_met(time_value, temp_op_info->met_unit());
                                                  // set string

      if (op_met_error) {
warning(w, "Illegal MET");
update_status("Illegal value for MET (correct value or Cancel): "
                    "time ::= digit {digit}", RING_BELL);
      }
    }
    op_period_error = read_time_value(period, time_value);
    temp_op_info->period(time_value);

    if (op_period_error) {
      warning(w, "Illegal Period");
      update_status("Illegal value for Period (correct value or Cancel): "
                  "time ::= digit {digit}", RING_BELL);
    }

    op_fw_error = read_time_value(fw, time_value);
    temp_op_info->fw(time_value);

    if (op_fw_error) {
      warning(w, "Illegal Finish Within");
      update_status("Illegal value for Finish Within time "
         "(correct value or Cancel): time ::= digit {digit}", RING_BELL);
    }

    break;
```

```
  case SPORADIC:

    if (!temp_op_info->is_terminator()) {
      op_met_error = read_time_value(met, time_value);
      temp_op_info->set_met(time_value, temp_op_info->met_unit());
                                                          // set string

      if (op_met_error) {
warning(w, "Illegal MET");
update_status("Illegal value for MET (correct value or Cancel): "
                    "time ::= digit {digit}", RING_BELL);
      }
    }

    op_mrt_error = read_time_value(mrt, time_value);
    temp_op_info->mrt(time_value);

    if (op_mrt_error) {
      warning(w, "Illegal MRT");
      update_status("Illegal value for MRT (correct value or Cancel): "
                  "time ::= digit {digit}", RING_BELL);
    }

    op_mcp_error = read_time_value(mcp, time_value);
    temp_op_info->mcp(time_value);

    if (op_mcp_error) {
      warning(w, "Illegal MCP");
      update_status("Illegal value for MCP (correct value or Cancel): "
                  "time ::= digit {digit}", RING_BELL);
    }

    break;
  }

  if ((temp_op_info->met() != UNDEFINED_TIME) &&
      reset_met_location)
    temp_op_info->set_default_met_location();
}


static void display_timing(int timing_type) {

  char *prompt, buffer[INPUT_LINE_SIZE];
  char *required_by_str = "Required By";

  // Use default MS if UNDEFINED_TIME
  if (temp_op_info->met()    == UNDEFINED_TIME)
    temp_op_info->met_unit(MS);
  if (temp_op_info->period() == UNDEFINED_TIME)
    temp_op_info->period_unit(MS);
  if (temp_op_info->fw()     == UNDEFINED_TIME)
    temp_op_info->fw_unit(MS);
  if (temp_op_info->mcp()    == UNDEFINED_TIME)
    temp_op_info->mcp_unit(MS);
  if (temp_op_info->mrt()    == UNDEFINED_TIME)
    temp_op_info->mrt_unit(MS);

  if (!met) {
    prompt = dup_str("MET:");
    met_label = XtVaCreateManagedWidget(prompt, xmLabelGadgetClass,
                                op_dialog,
    XmNx, TAB_LABEL,
    XmNy, ROW8,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);
    met =  XtVaCreateManagedWidget("met",
    xmTextFieldWidgetClass, op_dialog,
    XmNx, TAB_VALUE,
    XmNy, ROW8,
    XmNcolumns, 10,
    NULL);

    met_unit = time_unit_menu(op_dialog, "unit_box",
      temp_op_info->met_unit(), met_unit_cb,
      TAB_UNIT,ROW8);

    met_unit_label = XmOptionLabelGadget(met_unit);

    // requirements....
    met_req_by
      = XtVaCreateManagedWidget(required_by_str,
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB_REQMT,
XmNy, ROW8+3,
XmNwidth, 120,
NULL);

    XtAddCallback(met_req_by, XmNactivateCallback,
  req_by_cb, (XtPointer) temp_op_info->met_req_by_adr());

    req_by_label(met_req_by, *(temp_op_info->met_req_by_adr()));

    XtManageChild(met_label);
    XtManageChild(met);
    XtManageChild(met_unit);
    XtManageChild(met_req_by);

  }

  if (!period) {
    prompt = dup_str("Period:");
    period_label = XtVaCreateManagedWidget(prompt, xmLabelGadgetClass,
                                op_dialog,
    XmNx, TAB_LABEL,
```

```
    XmNy, ROW9,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);
    period =  XtVaCreateManagedWidget("period",
    xmTextFieldWidgetClass, op_dialog,
    XmNx, TAB_VALUE,
    XmNy, ROW9,
    XmNcolumns, 10,
    NULL);

    period_unit = time_unit_menu(op_dialog, "unit_box",
    temp_op_info->period_unit(), period_unit_cb,
    TAB_UNIT,ROW9);

    period_unit_label = XmOptionLabelGadget(period_unit);

    // requirements....
    period_req_by
      = XtVaCreateManagedWidget(required_by_str,
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB_REQMT,
XmNy, ROW9+3,
XmNwidth, 120,
NULL);

    XtAddCallback(period_req_by, XmNactivateCallback,
  req_by_cb, (XtPointer) temp_op_info->period_req_by_adr());

    req_by_label(period_req_by, *(temp_op_info->period_req_by_adr()));


    XtManageChild(period_label);
    XtManageChild(period);
    XtManageChild(period_unit);
    XtManageChild(period_req_by);

  }

  if (!mrt) {
    prompt = dup_str("MRT:");
    mrt_label = XtVaCreateManagedWidget(prompt, xmLabelGadgetClass,
                                op_dialog,
    XmNx, TAB_LABEL,
    XmNy, ROW10,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);
    mrt =  XtVaCreateManagedWidget("mrt",
    xmTextFieldWidgetClass, op_dialog,
    XmNx, TAB_VALUE,
    XmNy, ROW10,
    XmNcolumns, 10,
    NULL);

    mrt_unit = time_unit_menu(op_dialog, "unit_box",
    temp_op_info->mrt_unit(), mrt_unit_cb,
    TAB_UNIT,ROW10);

    mrt_unit_label = XmOptionLabelGadget(mrt_unit);

    // requirements....
    mrt_req_by
      = XtVaCreateManagedWidget(required_by_str,
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB_REQMT,
XmNy, ROW10+3,
XmNwidth, 120,
NULL);

    XtAddCallback(mrt_req_by, XmNactivateCallback,
  req_by_cb, (XtPointer) temp_op_info->mrt_req_by_adr());

    req_by_label(mrt_req_by, *(temp_op_info->mrt_req_by_adr()));

    // Not needed yet...
    XtUnmanageChild(mrt_label);
    XtUnmanageChild(mrt);
    XtUnmanageChild(mrt_unit);
    XtUnmanageChild(mrt_req_by);

  }

  if (!mcp) {
    prompt = dup_str("MCP:");
    mcp_label = XtVaCreateManagedWidget(prompt, xmLabelGadgetClass,
                                op_dialog,
    XmNx, TAB_LABEL,
    XmNy, ROW9,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);
    mcp =  XtVaCreateManagedWidget("mcp",
    xmTextFieldWidgetClass, op_dialog,
    XmNx, TAB_VALUE,
    XmNy, ROW9,
    XmNcolumns, 10,
    NULL);

    mcp_unit = time_unit_menu(op_dialog, "unit_box",
    temp_op_info->mcp_unit(), mcp_unit_cb,
    TAB_UNIT,ROW9);

    mcp_unit_label = XmOptionLabelGadget(mcp_unit);

    // requirements....
```

```
    mcp_req_by
      = XtVaCreateManagedWidget(required_by_str,
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB_REQMT,
XmNy, ROW9+3,
XmNwidth, 120,
NULL);

    XtAddCallback(mcp_req_by, XmNactivateCallback,
  req_by_cb, (XtPointer) temp_op_info->mcp_req_by_adr());

    req_by_label(mcp_req_by, *(temp_op_info->mcp_req_by_adr()));


    // Not needed yet...
    XtUnmanageChild(mcp_label);
    XtUnmanageChild(mcp);
    XtUnmanageChild(mcp_unit);
    XtUnmanageChild(mcp_req_by);

  }

  if (!fw) {
    prompt = dup_str("Finish");
    fw_label = XtVaCreateManagedWidget(prompt, xmLabelGadgetClass,
                          op_dialog,
    XmNx, TAB_LABEL,
    XmNy, ROW10,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);
    prompt = dup_str("Within:");
    fw_label2 = XtVaCreateManagedWidget(prompt, xmLabelGadgetClass,
                          op_dialog,
    XmNx, TAB_LABEL,
    XmNy, ROW10+15,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);
    fw =  XtVaCreateManagedWidget("fw",
    xmTextFieldWidgetClass, op_dialog,
    XmNx, TAB_VALUE,
    XmNy, ROW10,
    XmNcolumns, 10,
    NULL);

    fw_unit = time_unit_menu(op_dialog, "unit_box",
    temp_op_info->fw_unit(), fw_unit_cb,
    TAB_UNIT,ROW10);

    fw_unit_label = XmOptionLabelGadget(fw_unit);

    // requirements....
```

```
    fw_req_by
      = XtVaCreateManagedWidget(required_by_str,
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB_REQMT,
XmNy, ROW10+3,
XmNwidth, 120,
NULL);

    XtAddCallback(fw_req_by, XmNactivateCallback,
  req_by_cb, (XtPointer) temp_op_info->fw_req_by_adr());

    req_by_label(fw_req_by,*(temp_op_info->fw_req_by_adr()));


    XtManageChild(fw_label);   XtManageChild(fw_label2);
    XtManageChild(fw);
    XtManageChild(fw_unit);
    XtManageChild(fw_req_by);

  }

  switch (timing_type) {
  case NTC:               // None - all inactive

    if (!temp_op_info->is_terminator()) {
      temp_op_info->met(UNDEFINED_TIME);
      temp_op_info->met_unit(MS);
      temp_op_info->met_reqmts_dup(NULL);
    }
    temp_op_info->period(UNDEFINED_TIME);
    temp_op_info->period_unit(MS);
    temp_op_info->period_reqmts_dup(NULL);
    temp_op_info->fw(UNDEFINED_TIME);
    temp_op_info->fw_unit(MS);
    temp_op_info->fw_reqmts_dup(NULL);
    temp_op_info->mcp(UNDEFINED_TIME);
    temp_op_info->mcp_unit(MS);
    temp_op_info->mcp_reqmts_dup(NULL);
    temp_op_info->mrt(UNDEFINED_TIME);
    temp_op_info->mrt_unit(MS);
    temp_op_info->mrt_reqmts_dup(NULL);

    if (temp_op_info->is_terminator()) {
      sprintf(buffer, "%d", temp_op_info->met());
      XtVaSetValues(met, XmNvalue, buffer, NULL);
    }
    else {
      XtVaSetValues(met, XmNvalue, "", NULL);
    }
    req_by_label(met_req_by, *(temp_op_info->met_req_by_adr()));
    XtVaSetValues(period, XmNvalue, "", NULL);
    req_by_label(period_req_by, NULL);
    XtVaSetValues(fw,     XmNvalue, "", NULL);
```

```
  req_by_label(fw_req_by, NULL);
  XtVaSetValues(mcp,    XmNvalue, "", NULL);
  req_by_label(mcp_req_by, NULL);
  XtVaSetValues(mrt,    XmNvalue, "", NULL);
  req_by_label(mrt_req_by, NULL);

  XtVaSetValues(met_label,    XmNsensitive, False, NULL);
  XtVaSetValues(met,          XmNsensitive, False, NULL);
  XtVaSetValues(met_unit,     XmNsensitive, False, NULL);
  XtVaSetValues(met_unit_label, XmNsensitive, False, NULL);
  if (!temp_op_info->is_terminator())
    XtVaSetValues(met_req_by, XmNsensitive, False, NULL);
  else
    XtVaSetValues(met_req_by, XmNsensitive, True,  NULL);
  XtVaSetValues(period_label, XmNsensitive, False, NULL);
  XtVaSetValues(period,       XmNsensitive, False, NULL);
  XtVaSetValues(period_unit,  XmNsensitive, False, NULL);
  XtVaSetValues(period_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(period_req_by,XmNsensitive, False, NULL);
  XtVaSetValues(fw_label,     XmNsensitive, False, NULL);
  XtVaSetValues(fw_label2,    XmNsensitive, False, NULL);
  XtVaSetValues(fw,           XmNsensitive, False, NULL);
  XtVaSetValues(fw_unit,      XmNsensitive, False, NULL);
  XtVaSetValues(fw_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(fw_req_by,    XmNsensitive, False, NULL);
  XtVaSetValues(mrt_label,    XmNsensitive, False, NULL);
  XtVaSetValues(mrt,          XmNsensitive, False, NULL);
  XtVaSetValues(mrt_unit,     XmNsensitive, False, NULL);
  XtVaSetValues(mrt_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(mrt_req_by,   XmNsensitive, False, NULL);
  XtVaSetValues(mcp_label,    XmNsensitive, False, NULL);
  XtVaSetValues(mcp,          XmNsensitive, False, NULL);
  XtVaSetValues(mcp_unit,     XmNsensitive, False, NULL);
  XtVaSetValues(mcp_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(mcp_req_by,   XmNsensitive, False, NULL);

  break;

case PERIODIC:          // Met, Period, Finish Within

  if (temp_op_info->met() != UNDEFINED_TIME) {
    sprintf(buffer, "%d", temp_op_info->met());
    XtVaSetValues(met, XmNvalue, buffer, NULL);
  }
  else {
    XtVaSetValues(met, XmNvalue, "", NULL);
  }
  if (temp_op_info->period() != UNDEFINED_TIME) {
    sprintf(buffer, "%d", temp_op_info->period());
    XtVaSetValues(period, XmNvalue, buffer, NULL);
  }
  else {
    XtVaSetValues(period, XmNvalue, "", NULL);
  }
  if (temp_op_info->fw() != UNDEFINED_TIME) {
    sprintf(buffer, "%d", temp_op_info->fw());
    XtVaSetValues(fw, XmNvalue, buffer, NULL);
  }
  else {
    XtVaSetValues(fw, XmNvalue, "", NULL);
  }

  if (!temp_op_info->is_terminator()) {
    XtVaSetValues(met_label,    XmNsensitive, True,  NULL);
    XtVaSetValues(met,          XmNsensitive, True,  NULL);
    XtVaSetValues(met_unit,     XmNsensitive, True,  NULL);
    XtVaSetValues(met_unit_label, XmNsensitive, True,  NULL);
  }
  else {
    XtVaSetValues(met_label,    XmNsensitive, False,  NULL);
    XtVaSetValues(met,          XmNsensitive, False,  NULL);
    XtVaSetValues(met_unit,     XmNsensitive, False,  NULL);
    XtVaSetValues(met_unit_label, XmNsensitive, False,  NULL);
  }
  XtVaSetValues(met_req_by,   XmNsensitive, True,  NULL);

  XtVaSetValues(period_label, XmNsensitive, True,  NULL);
  XtVaSetValues(period,       XmNsensitive, True,  NULL);
  XtVaSetValues(period_unit,  XmNsensitive, True,  NULL);
  XtVaSetValues(period_unit_label, XmNsensitive, True,  NULL);
  XtVaSetValues(period_req_by,XmNsensitive, True,  NULL);
  XtVaSetValues(fw_label,     XmNsensitive, True,  NULL);
  XtVaSetValues(fw_label2,    XmNsensitive, True,  NULL);
  XtVaSetValues(fw,           XmNsensitive, True,  NULL);
  XtVaSetValues(fw_unit,      XmNsensitive, True,  NULL);
  XtVaSetValues(fw_unit_label, XmNsensitive, True,  NULL);
  XtVaSetValues(fw_req_by,    XmNsensitive, True,  NULL);
  XtVaSetValues(mrt_label,    XmNsensitive, False, NULL);
  XtVaSetValues(mrt,          XmNsensitive, False, NULL);
  XtVaSetValues(mrt_unit,     XmNsensitive, False, NULL);
  XtVaSetValues(mrt_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(mrt_req_by,   XmNsensitive, False, NULL);
  XtVaSetValues(mcp_label,    XmNsensitive, False, NULL);
  XtVaSetValues(mcp,          XmNsensitive, False, NULL);
  XtVaSetValues(mcp_unit,     XmNsensitive, False, NULL);
  XtVaSetValues(mcp_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(mcp_req_by,   XmNsensitive, False, NULL);

  XtUnmanageChild(mrt_label);
  XtUnmanageChild(mrt);
  XtUnmanageChild(mrt_unit);
  XtUnmanageChild(mrt_req_by);
  XtUnmanageChild(mcp_label);
  XtUnmanageChild(mcp);
  XtUnmanageChild(mcp_unit);
  XtUnmanageChild(mcp_req_by);
```

```
  XtManageChild(met_label);
  XtManageChild(met);
  XtManageChild(met_unit);
  XtManageChild(met_req_by);
  XtManageChild(period_label);
  XtManageChild(period);
  XtManageChild(period_unit);
  XtManageChild(period_req_by);
  XtManageChild(fw_label);     XtManageChild(fw_label2);
  XtManageChild(fw);
  XtManageChild(fw_unit);
  XtManageChild(fw_req_by);

  break;

case SPORADIC:          // Met, MRT, MCP

  if (temp_op_info->met() != UNDEFINED_TIME) {
    sprintf(buffer, "%d", temp_op_info->met());
    XtVaSetValues(met, XmNvalue, buffer, NULL);
  }
  else {
    XtVaSetValues(met, XmNvalue, "", NULL);
  }
  if (temp_op_info->mcp() != UNDEFINED_TIME) {
    sprintf(buffer, "%d", temp_op_info->mcp());
    XtVaSetValues(mcp, XmNvalue, buffer, NULL);
  }
  else {
    XtVaSetValues(mcp, XmNvalue, "", NULL);
  }
  if (temp_op_info->mrt() != UNDEFINED_TIME) {
    sprintf(buffer, "%d", temp_op_info->mrt());
    XtVaSetValues(mrt, XmNvalue, buffer, NULL);
  }
  else {
    XtVaSetValues(mrt, XmNvalue, "", NULL);
  }

  if (!temp_op_info->is_terminator()) {
    XtVaSetValues(met_label,    XmNsensitive, True,  NULL);
    XtVaSetValues(met,          XmNsensitive, True,  NULL);
    XtVaSetValues(met_unit,     XmNsensitive, True,  NULL);
    XtVaSetValues(met_unit_label, XmNsensitive, True,  NULL);
  }
  else {
    XtVaSetValues(met_label,    XmNsensitive, False,  NULL);
    XtVaSetValues(met,          XmNsensitive, False,  NULL);
    XtVaSetValues(met_unit,     XmNsensitive, False,  NULL);
    XtVaSetValues(met_unit_label, XmNsensitive, False,  NULL);
  }
  XtVaSetValues(met_req_by,   XmNsensitive, True,  NULL);
  XtVaSetValues(period_label, XmNsensitive, False, NULL);
  XtVaSetValues(period,       XmNsensitive, False, NULL);
  XtVaSetValues(period_unit,  XmNsensitive, False, NULL);
  XtVaSetValues(period_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(period_req_by,XmNsensitive, False, NULL);
  XtVaSetValues(fw_label,     XmNsensitive, False, NULL);
  XtVaSetValues(fw_label2,    XmNsensitive, False, NULL);
  XtVaSetValues(fw,           XmNsensitive, False, NULL);
  XtVaSetValues(fw_unit,      XmNsensitive, False, NULL);
  XtVaSetValues(fw_unit_label, XmNsensitive, False, NULL);
  XtVaSetValues(fw_req_by,    XmNsensitive, False, NULL);
  XtVaSetValues(mrt_label,    XmNsensitive, True,  NULL);
  XtVaSetValues(mrt,          XmNsensitive, True,  NULL);
  XtVaSetValues(mrt_unit,     XmNsensitive, True,  NULL);
  XtVaSetValues(mrt_unit_label, XmNsensitive, True,  NULL);
  XtVaSetValues(mrt_req_by,   XmNsensitive, True,  NULL);
  XtVaSetValues(mcp_label,    XmNsensitive, True,  NULL);
  XtVaSetValues(mcp,          XmNsensitive, True,  NULL);
  XtVaSetValues(mcp_unit,     XmNsensitive, True,  NULL);
  XtVaSetValues(mcp_unit_label, XmNsensitive, True,  NULL);
  XtVaSetValues(mcp_req_by,   XmNsensitive, True,  NULL);

  XtManageChild(met_label);
  XtManageChild(met);
  XtManageChild(met_unit);
  XtManageChild(met_req_by);
  XtManageChild(mrt_label);
  XtManageChild(mrt);
  XtManageChild(mrt_unit);
  XtManageChild(mrt_req_by);
  XtManageChild(mcp_label);
  XtManageChild(mcp);
  XtManageChild(mcp_unit);
  XtManageChild(mcp_req_by);


  XtUnmanageChild(period_label);
  XtUnmanageChild(period);
  XtUnmanageChild(period_unit);
  XtUnmanageChild(period_req_by);
  XtUnmanageChild(fw_label);     XtUnmanageChild(fw_label2);
  XtUnmanageChild(fw);
  XtUnmanageChild(fw_unit);
  XtUnmanageChild(fw_req_by);


  break;
 }
}

static void timing_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
```

```
    (XmToggleButtonCallbackStruct *) cbs;

  if (state->set)
      temp_op_info->timing_type( (int) which );

  display_timing(temp_op_info->timing_type());

  return;
}

static void output_guard_ok_pushed(Widget w, XtPointer client_data,
   XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);

  temp_op_info->output_guard_list(text);

  if ((text != NULL) && (*text != '\0')) {
    if (!output_guard_value) {
      output_guard_value =  XtVaCreateManagedWidget("output_guard_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB3,
XmNy, ROW11,
XmNwidth, 250,
                                XmNeditable, False,
                                XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
    }
    copy_str_eps(text, &output_guard_prefix,38);
    XtVaSetValues(output_guard_value, XmNvalue, output_guard_prefix, NULL);
    free(output_guard_prefix);  output_guard_prefix = NULL;
  }
  else {
    if (output_guard_value) {
      XtDestroyWidget(output_guard_value);
      output_guard_value = NULL;
    }
  }

  free(text);              text     = NULL;

  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}

static void output_guard_cb(Widget w, XtPointer client_data,
    XtPointer call_data) {
```

```
Widget    dialog, pane, rc, text_w, action_a;
XmString  string;
char      *description;

static ActionAreaItem   action_items[] = {
  {"OK",     output_guard_ok_pushed, NULL                    },
  {"Cancel", close_dialog,           NULL                   },
  {"Help",   help_cb,                "output_guard.hlp"  }
};

dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
XtParent(w),
XmNtitle, "Operator Output Guard",
XmNdeleteResponse, XmDESTROY,
NULL);

action_items[1].data = (XtPointer)dialog;//Set cancel buttons client_data

pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
XmNsashWidth,   1,
XmNsashHeight,  1,
NULL);

rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                      pane, NULL);
string = XmStringCreateSimple("View or Edit Operator "
                              "Output Guard Equation");
XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
XmNlabelString, string,
NULL);
XmStringFree(string);

description = temp_op_info->output_guard_list();

int     n = 0;
Arg     args[10];
XtSetArg(args[n], XmNrows,                  12); n++;
XtSetArg(args[n], XmNcolumns,               70); n++;
XtSetArg(args[n], XmNscrollVertical,        true); n++;
XtSetArg(args[n], XmNscrollHorizontal,      true); n++;
XtSetArg(args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
XtSetArg(args[n], XmNeditable,              true); n++;
XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
XtSetArg(args[n], XmNwordWrap,              true); n++;
XtSetArg(args[n], XmNvalue,                 description); n++;
text_w = XmCreateScrolledText(rc, "text-field", args, n);
XtManageChild(text_w);
//text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
//      rc, NULL);

XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

XtManageChild(rc);
```

```
  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);

}

static void trigger_if_cond_ok_pushed(Widget w, XtPointer client_data,
      XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);

  temp_op_info->trigger_if_condition(text);

  if ((text != NULL) && (*text != '\0')) {
    if (!trigger_if_value) {
      trigger_if_value =  XtVaCreateManagedWidget("trigger_if_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB5,
XmNy, ROW5,
XmNwidth, 170,
                                XmNeditable, False,
                                XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
    }
    copy_str_eps(text, &if_condition_prefix,25);
    XtVaSetValues(trigger_if_value, XmNvalue, if_condition_prefix, NULL);
    free(if_condition_prefix);     if_condition_prefix = NULL;
  }
  else {
    if (trigger_if_value) {
      XtDestroyWidget(trigger_if_value);
      trigger_if_value = NULL;
    }
  }

  free(text);              text     = NULL;

  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}
```

```
static void trigger_if_cond_cb(Widget w, XtPointer client_data,
    XtPointer call_data) {

  Widget    dialog, pane, rc, text_w, action_a;
  XmString  string;
  char      *description;

  static ActionAreaItem   action_items[] = {
    {"OK",     trigger_if_cond_ok_pushed, NULL                    },
    {"Cancel", close_dialog,          NULL                    },
    {"Help",   help_cb,               "trigger_if_cond.hlp"  }
  };

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 XtParent(w),
 XmNtitle, "Operator Trigger If Condition",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog; //Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                        pane, NULL);
  string = XmStringCreateSimple("View or Edit Operator Trigger "
                                "If Condition");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  description = temp_op_info->trigger_if_condition();

  int     n = 0;
  Arg     args[10];
  XtSetArg(args[n], XmNrows,                 12); n++;
  XtSetArg(args[n], XmNcolumns,              70); n++;
  XtSetArg(args[n], XmNscrollVertical,       true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,     true); n++;
  XtSetArg(args[n], XmNeditMode,             XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,             true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,             true); n++;
  XtSetArg(args[n], XmNvalue,                description); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);
  //text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
  //      rc, NULL);
```

```
  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);

}

static void exception_ok_pushed(Widget w, XtPointer client_data,
      XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);

  temp_op_info->exception_list(text);

  if ((text != NULL) && (*text != '\0')) {
    if (!exception_value) {
      exception_value =  XtVaCreateManagedWidget("exception_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB3,
XmNy, ROW12,
XmNwidth, 250,
                                XmNeditable, False,
                                XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
    }
    copy_str_eps(text, &exception_prefix,38);
    XtVaSetValues(exception_value, XmNvalue, exception_prefix, NULL);
    free(exception_prefix);     exception_prefix = NULL;
  }
  else {
    if (exception_value) {
      XtDestroyWidget(exception_value);
      exception_value = NULL;
    }
  }

  free(text);              text     = NULL;
  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}

static void exception_cb(Widget w, XtPointer client_data,
    XtPointer call_data) {

  Widget    dialog, pane, rc, text_w, action_a;
  XmString  string;
  char      *description;

  static ActionAreaItem   action_items[] = {
    {"OK",     exception_ok_pushed,  NULL                    },
    {"Cancel", close_dialog,         NULL                    },
    {"Help",   help_cb,              "exceptions_list.hlp"   }
  };

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 XtParent(w),
 XmNtitle, "Operator Exceptions",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog;//Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                        pane, NULL);
  string = XmStringCreateSimple("View or Edit Operator Exceptions");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  description = temp_op_info->exception_list();

  int     n = 0;
  Arg     args[10];
  XtSetArg(args[n], XmNrows,                  12); n++;
  XtSetArg(args[n], XmNcolumns,               70); n++;
  XtSetArg(args[n], XmNscrollVertical,        true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,      true); n++;
  XtSetArg(args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,              true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,              true); n++;
  XtSetArg(args[n], XmNvalue,                 description); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
```

```
  XtManageChild(text_w);
  //text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
  //      rc, NULL);

  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);

}


static void timer_op_ok_pushed(Widget w, XtPointer client_data,
      XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);

  temp_op_info->timer_op_list(text);

  if ((text != NULL) && (*text != '\0')) {
    if (!timer_op_value) {
      timer_op_value =  XtVaCreateManagedWidget("timer_op_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB3,
XmNy, ROW13,
XmNwidth, 250,
                              XmNeditable, False,
                              XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
    }
    copy_str_eps(text, &timer_op_prefix,38);
    XtVaSetValues(timer_op_value, XmNvalue, timer_op_prefix, NULL);
    free(timer_op_prefix);    timer_op_prefix = NULL;
  }
  else {
    if (timer_op_value) {
      XtDestroyWidget(timer_op_value);
      timer_op_value = NULL;
    }
  }

  free(text);              text     = NULL;

  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}

static void timer_op_cb(Widget w, XtPointer client_data,
    XtPointer call_data) {

  Widget    dialog, pane, rc, text_w, action_a;
  XmString  string;
  char      *description;

  static ActionAreaItem   action_items[] = {
    {"OK",     timer_op_ok_pushed, NULL                    },
    {"Cancel", close_dialog,         NULL                     },
    {"Help",   help_cb,              "timer_list.hlp"   }
  };

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 XtParent(w),
 XmNtitle, "Operator Timers",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog;//Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                        pane, NULL);
  string = XmStringCreateSimple("View or Edit Operator Timers");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  description = temp_op_info->timer_op_list();

  int     n = 0;
  Arg     args[10];
  XtSetArg(args[n], XmNrows,                  12); n++;
  XtSetArg(args[n], XmNcolumns,               70); n++;
  XtSetArg(args[n], XmNscrollVertical,        true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,      true); n++;
  XtSetArg(args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,              true); n++;
```

```
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,              true); n++;
  XtSetArg(args[n], XmNvalue,                 description); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);
  //text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
  //      rc, NULL);

  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);

}


static void operator_ok_cb(Widget w, XtPointer client_data,
                           XtPointer call_data) {

  OperatorObject *op_being_updated;
  op_being_updated = (OperatorObject *) client_data;

//  Check for any errors
  if (op_name_error)   return;
  if (op_met_error)    return;
  if (op_period_error) return;
  if (op_fw_error)     return;
  if (op_mcp_error)    return;
  if (op_mrt_error)    return;

  //  op_being_updated->select();
  op_being_updated->erase();

  *op_being_updated = *temp_op_info;

  op_being_updated->draw(SOLID);
  //  op_being_updated->unselect();

  XtDestroyWidget(XtParent(w));

  // Recover any allocated memory
  delete temp_op_info;

  op_being_updated->set_modified();
  save_state(SAVE_REQUIRED);
  clear_status();

  return;
}


static void operator_cancel_cb(Widget w, XtPointer client_data,
                               XtPointer call_data) {

  XtDestroyWidget(XtParent(w));

  // Recover any allocated memory
  delete temp_op_info;

  clear_status();

  return;
}


static void operator_help_cb(Widget w, XtPointer client_data,
                             XtPointer call_data) {
  help_cb(w, client_data, call_data);

  return;
}


void operator_property_dialog(Widget parent,
                              OperatorObject *op_being_updated,
     int x, int y, BOOLEAN parent_terminator,
     ID_LIST avail_impl_langs,
     GraphObjectList *graphic_list) {

  XmString tmp;

  Arg   args[10];
  char *prompt, buffer[INPUT_LINE_SIZE];
  int   ac;

  opDialogGraphicList = graphic_list;

  output_guard_value  = NULL;
  exception_value     = NULL;
  timer_op_value      = NULL;
  if_condition_prefix = NULL;
  output_guard_prefix = NULL;
  exception_prefix    = NULL;
  timer_op_prefix     = NULL;

  Global_avail_impl_langs = avail_impl_langs;
```

```
// reset error status
clear_status();
op_name_error     = false;
op_met_error      = false;
op_period_error   = false;
op_mrt_error      = false;
op_mcp_error      = false;
op_fw_error       = false;

met    = NULL;
period = NULL;
mrt    = NULL;
mcp    = NULL;
fw     = NULL;

temp_op_info = new OperatorObject();

if (op_being_updated == (OperatorObject *)NULL) {
  prompt = dup_str("Bad Operator Pointer Passed To Operator Dialog");
  warning(parent, prompt);
  free(prompt);         prompt = NULL;
} else {  // Build dialog

  // Get data from selected operator and store locally
  *temp_op_info = *op_being_updated;

  ac = 0;
  XtSetArg(args[ac], XmNheight,       WIN_HEIGHT);   ac++;
  XtSetArg(args[ac], XmNwidth,        WIN_WIDTH);    ac++;
  XtSetArg(args[ac], XmNautoUnmanage, False);        ac++;
  tmp = XmStringCreateSimple("Operator Property");
  XtSetArg(args[ac], XmNdialogTitle,  tmp);          ac++;

  op_dialog = XmCreateBulletinBoardDialog(GetTopShell(parent),
    "Operator_Property", args, ac);
  XmStringFree(tmp);


  if (parent_terminator) {
    tmp = XmStringCreateSimple("Terminator");
    oper_label = XtVaCreateManagedWidget("oper_label",
                        xmLabelGadgetClass, op_dialog,
  XmNx, TAB6,
  XmNy, ROW2,
  XmNalignment, XmALIGNMENT_BEGINNING,
  XmNlabelType, XmSTRING,
  XmNlabelString, tmp,
  NULL);
    XmStringFree(tmp);
  }
  else {
    XmString oper_string, term_string;
    oper_string = XmStringCreateSimple("Operator");
    term_string = XmStringCreateSimple("Terminator");

    oper_label = XmVaCreateSimpleOptionMenu(op_dialog, "oper_label",
NULL, NULL, temp_op_info->is_terminator(), operator_cb,
XmVaPUSHBUTTON, oper_string, NULL, NULL, NULL,
        XmVaPUSHBUTTON, term_string, NULL, NULL, NULL,
        XmNx, TAB6,
        XmNy, ROW2,
        NULL);

    XmStringFree(term_string);
    XmStringFree(oper_string);
    XtManageChild(oper_label);
  }

  /*********************  Name  ***************************/

  char *name_str;
  prompt = dup_str("Name:");
  XtVaCreateManagedWidget(prompt,
  xmLabelGadgetClass, op_dialog,
  XmNx, TAB2,
  XmNy, ROW2,
  NULL);
  free(prompt);       prompt = NULL;

  oper_name =  XtVaCreateManagedWidget("oper_name",
  xmTextFieldWidgetClass, op_dialog,
  XmNx, TAB_VALUE,
  XmNy, ROW2,
  NULL);
  name_str = temp_op_info->name();
  XtVaSetValues(oper_name, XmNvalue, name_str, NULL);
  free(name_str);    name_str = NULL;


  /*********************  Implementation  *****************/

  prompt = dup_str("Implementation:");
  XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, op_dialog,
  XmNx, TAB2,
  XmNy, ROW3,
  XmNalignment, XmALIGNMENT_BEGINNING,
  NULL);
  free(prompt);       prompt = NULL;

  ID_LIST  langPtr;
  int      langCount, n, i;
  XmString langStr;
```

```
    MenuItem *impl_lang_options;

    langPtr = avail_impl_langs;
    langCount = 0;
    while (langPtr != NULL) {
      langCount++;
      langPtr = langPtr->next;
    }

    impl_lang_options  = (MenuItem *) malloc((langCount+1) *
                                        sizeof(MenuItem));

    n = 0;
    langPtr = avail_impl_langs;
    while (langPtr) {
      (impl_lang_options+n)->label         = langPtr->id;
      (impl_lang_options+n)->widget_class  = &xmPushButtonGadgetClass;
      (impl_lang_options+n)->sensitive     = true;
      (impl_lang_options+n)->act_display   =
(strcmp(langPtr->id,
       *temp_op_info->operator_impl_lang_adr()) == 0) ? true : false;
      (impl_lang_options+n)->mnemonic      = NULL;
      (impl_lang_options+n)->accelerator   = NULL;
      (impl_lang_options+n)->accel_text    = NULL;
      (impl_lang_options+n)->callback      = impl_lang_cb;
      (impl_lang_options+n)->callback_data = (XtPointer) n;
      (impl_lang_options+n)->subitems      = NULL;
      n++;
      langPtr = langPtr->next;
    }
    (impl_lang_options+n)->label = NULL;    // terminate list

    impl_lang = BuildMenu(op_dialog, XmMENU_OPTION,
  "Implementation Language:        ",
  NULL, impl_lang_options);

    XtVaSetValues(impl_lang, XmNx, TAB2-3, NULL);
    XtVaSetValues(impl_lang, XmNy, ROW3, NULL);
    /*
    XtAddCallback(impl_lang, XmNactivateCallback,
  impl_lang_cb,
  (XtPointer) temp_op_info->operator_impl_lang_adr());
    */
    free((char*) impl_lang_options);
    XtManageChild(impl_lang);


    /*********************  Trigger   ************************/

    prompt = dup_str("Trigger:");
    XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, op_dialog,
    XmNx, TAB2,
    XmNy, ROW4,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);      prompt = NULL;

    XmString by_some, by_all, unprotected;

    unprotected  = XmStringCreateSimple("unprotected");
    by_some      = XmStringCreateSimple("By Some");
    by_all       = XmStringCreateSimple("By All");

    trigger = XmVaCreateSimpleOptionMenu(op_dialog, "trigger",
           NULL, NULL, op_being_updated->trigger_type(), trigger_cb,
           XmVaPUSHBUTTON, unprotected,  NULL, NULL, NULL,
           XmVaPUSHBUTTON, by_some,      NULL, NULL, NULL,
           XmVaPUSHBUTTON, by_all,       NULL, NULL, NULL,
           XmNx, TAB_VALUE-10,
           XmNy, ROW4,
           NULL);

    XmStringFree(by_all);
    XmStringFree(by_some);
    XmStringFree(unprotected);

    if (op_being_updated->trigger_type() != UNPROTECTED) {
      trigger_list = XtVaCreateManagedWidget(dup_str("Stream List"),
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB5,
XmNy, ROW4+3,
XmNwidth, 170,
NULL);
      XtAddCallback(trigger_list, XmNactivateCallback,
    trigger_list_cb,
    (XtPointer) temp_op_info->trigger_set_adr());

      trigger_label(trigger_list, *(temp_op_info->trigger_set_adr()));
    }
    else
      trigger_list = NULL;

    // if condition
    char* if_cond_str = NULL;

    trigger_if_cond
      = XtVaCreateManagedWidget(dup_str("If Condition"),
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB_VALUE,
XmNy, ROW5,
XmNwidth, 100,
NULL);

    XtAddCallback(trigger_if_cond, XmNactivateCallback,
  trigger_if_cond_cb, (XtPointer)NULL);
```

```
    if_cond_str = temp_op_info->trigger_if_condition();

    if ((if_cond_str != NULL) && (*if_cond_str != '\0')) {
      trigger_if_value =  XtVaCreateManagedWidget("trigger_if_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB5,
XmNy, ROW5,
XmNwidth, 170,
                              XmNeditable, False,
                              XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
      copy_str_eps(if_cond_str,&if_condition_prefix,25);
      XtVaSetValues(trigger_if_value, XmNvalue,
           if_condition_prefix, NULL);
      free(if_condition_prefix);  if_condition_prefix = NULL;
    }
    else
      trigger_if_value = NULL;

    free(if_cond_str);       if_cond_str = NULL;

    // requirements....
    prompt = dup_str("Required By");
    trigger_req_by
      = XtVaCreateManagedWidget(prompt,
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB_VALUE,
XmNy, ROW6,
XmNwidth, 120,
NULL);
    free(prompt);      prompt = NULL;

    XtAddCallback(trigger_req_by, XmNactivateCallback,
   req_by_cb, (XtPointer) temp_op_info->trigger_req_by_adr());

    req_by_label(trigger_req_by, *(temp_op_info->trigger_req_by_adr()));

    XtManageChild(trigger_req_by);
    XtManageChild(trigger);

    /*********************  Timing    ***********************/

    prompt = dup_str("Timing:");
    XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, op_dialog,
    XmNx, TAB2,
    XmNy, ROW7,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);      prompt = NULL;

    XmString ntc, periodic, sporadic;

    ntc      = XmStringCreateSimple("Non-Time Critical");
    periodic = XmStringCreateSimple("Periodic");
    sporadic = XmStringCreateSimple("Sporadic");

    timing = XmVaCreateSimpleOptionMenu(op_dialog, "timing",
           NULL, NULL, op_being_updated->timing_type(), timing_cb,
           XmVaPUSHBUTTON, ntc,      NULL, NULL, NULL,
           XmVaPUSHBUTTON, periodic, NULL, NULL, NULL,
           XmVaPUSHBUTTON, sporadic, NULL, NULL, NULL,
           XmNx, TAB_VALUE-10,
           XmNy, ROW7,
           NULL);

    XmStringFree(sporadic);
    XmStringFree(periodic);
    XmStringFree(ntc);

    XtManageChild(timing);

    display_timing(op_being_updated->timing_type());

    /*********************  output guard  ************************/

    char* output_guard_str = NULL;

    output_guard
      = XtVaCreateManagedWidget(dup_str("Output Guards"),
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB2,
XmNy, ROW11,
XmNwidth, 100,
NULL);

    XtAddCallback(output_guard, XmNactivateCallback,
  output_guard_cb, (XtPointer)NULL);

    output_guard_str = temp_op_info->output_guard_list();

    if ((output_guard_str != NULL) && (*output_guard_str != '\0')) {
      output_guard_value =  XtVaCreateManagedWidget("output_guard_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB3,
XmNy, ROW11,
XmNwidth, 250,
                              XmNeditable, False,
                              XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
      copy_str_eps(output_guard_str,&output_guard_prefix,38);
      XtVaSetValues(output_guard_value, XmNvalue,
           output_guard_prefix, NULL);
      free(output_guard_prefix);    output_guard_prefix = NULL;
    }
```

```
    else
      output_guard_value = NULL;

    free(output_guard_str);       output_guard_str = NULL;

    /*******************  exceptions  ************************/

    char* exception_str = NULL;

    exception
      = XtVaCreateManagedWidget(dup_str("Exceptions"),
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB2,
XmNy, ROW12,
XmNwidth, 100,
NULL);

    XtAddCallback(exception, XmNactivateCallback,
  exception_cb, (XtPointer)NULL);

    exception_str = temp_op_info->exception_list();

    if ((exception_str != NULL) && (*exception_str != '\0')) {
      exception_value =  XtVaCreateManagedWidget("exception_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB3,
XmNy, ROW12,
XmNwidth, 250,
                              XmNeditable, False,
                              XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
      copy_str_eps(exception_str,&exception_prefix,38);
      XtVaSetValues(exception_value, XmNvalue,
           exception_prefix, NULL);
      free(exception_prefix);   exception_prefix = NULL;
    }
    else
      exception_value = NULL;

    free(exception_str);   exception_str = NULL;

    /*******************  timer ops  ************************/

    char* timer_str = NULL;

    timer_op
      = XtVaCreateManagedWidget(dup_str("Timer Ops"),
xmPushButtonGadgetClass, op_dialog,
XmNx, TAB2,
XmNy, ROW13,
XmNwidth, 100,
NULL);
```

```
    XtAddCallback(timer_op, XmNactivateCallback,
  timer_op_cb, (XtPointer)NULL);

    timer_str = temp_op_info->timer_op_list();

    if ((timer_str != NULL) && (*timer_str != '\0')) {
      timer_op_value =  XtVaCreateManagedWidget("timer_op_value",
xmTextFieldWidgetClass, op_dialog,
XmNx, TAB3,
XmNy, ROW13,
XmNwidth, 250,
                              XmNeditable, False,
                              XmNcursorPositionVisible, False,
          XmNshadowThickness, 1,
NULL);
      copy_str_eps(timer_str,&timer_op_prefix,38);
      XtVaSetValues(timer_op_value, XmNvalue,
           timer_op_prefix, NULL);
      free(timer_op_prefix);    timer_op_prefix = NULL;
    }
    else
      timer_op_value = NULL;

    free(timer_str);      timer_str = NULL;



    /******************* Spec Buttons ************************/

    int but_spacing;

    XtVaCreateManagedWidget("separator", xmSeparatorWidgetClass,
                            op_dialog,
    XmNy,     ROW14 - 8,
    XmNwidth, WIN_WIDTH,
    NULL);

    but_spacing = (WIN_WIDTH - (3 * 110))/4;

    ac = 0;
    XtSetArg(args[ac], XmNx, but_spacing+10); ac++;
    XtSetArg(args[ac], XmNy, ROW14); ac++;
    XtSetArg(args[ac], XmNwidth, 110);          ac++;
    spec_keyword_button =
      XmCreatePushButton(op_dialog, "    Keywords    ", args, ac);
    XtManageChild(spec_keyword_button);
    keyword_label(spec_keyword_button, *temp_op_info->key_word_list_adr());

    ac = 0;
    XtSetArg(args[ac], XmNx, 2*but_spacing+10 + 110); ac++;
    XtSetArg(args[ac], XmNy, ROW14); ac++;
    XtSetArg(args[ac], XmNwidth, 110);          ac++;
```

```
    spec_informal_button =
      XmCreatePushButton(op_dialog, " Informal Desc  ", args, ac);
    XtManageChild(spec_informal_button);
    informal_label(spec_informal_button,
   temp_op_info->operator_informal_desc_adr());

    ac = 0;
    XtSetArg(args[ac], XmNx, 3*but_spacing+10 + 220); ac++;
    XtSetArg(args[ac], XmNy, ROW14); ac++;
    XtSetArg(args[ac], XmNwidth, 110);          ac++;
    spec_formal_button =
      XmCreatePushButton(op_dialog, "  Formal Desc   ", args, ac);
    XtManageChild(spec_formal_button);
    formal_label(spec_formal_button,
temp_op_info->operator_formal_desc_adr());


    XtAddCallback(spec_keyword_button, XmNactivateCallback,
  keyword_cb,
  (XtPointer) temp_op_info->key_word_list_adr());

    XtAddCallback(spec_informal_button, XmNactivateCallback,
  informal_desc_cb,
  (XtPointer) temp_op_info->operator_informal_desc_adr());

    XtAddCallback(spec_formal_button, XmNactivateCallback,
  formal_desc_cb,
  (XtPointer) temp_op_info->operator_formal_desc_adr());



    /*******************  Action Buttons  **********************/

    XtVaCreateManagedWidget("separator", xmSeparatorWidgetClass, op_dialog,
    XmNy,     ROW15 - 8,
    XmNwidth, WIN_WIDTH,
    NULL);


    // create the OK, Cancel, Help buttons
     ,
    but_spacing = (WIN_WIDTH - (3 * 80))/4;

    ac = 0;
    XtSetArg(args[ac], XmNx,     but_spacing+10); ac++;     // 20, 125, 230
    XtSetArg(args[ac], XmNy,     ROW15);       ac++;
    XtSetArg(args[ac], XmNwidth, 80);          ac++;
    XtSetArg(args[ac], XmNlabelString,
     XmStringCreateSimple("OK"));
    ok_button = XmCreatePushButton(op_dialog, "   OK   ", args, ac);
    XtManageChild(ok_button);

    ac = 0;
    XtSetArg(args[ac], XmNx, 2*but_spacing+10 + 80); ac++;
    XtSetArg(args[ac], XmNy, ROW15);              ac++;
    XtSetArg(args[ac], XmNwidth, 80);             ac++;
    XtSetArg(args[ac], XmNlabelString,
     XmStringCreateSimple("Cancel"));
    cancel_button = XmCreatePushButton(op_dialog, " Cancel ", args, ac);
    XtManageChild(cancel_button);

    ac = 0;
    XtSetArg(args[ac], XmNx, 3*but_spacing+10 + 160); ac++;
    XtSetArg(args[ac], XmNy, ROW15);               ac++;
    XtSetArg(args[ac], XmNwidth, 80);              ac++;
    XtSetArg(args[ac], XmNlabelString,
     XmStringCreateSimple("Cancel"));
    help_button = XmCreatePushButton(op_dialog, "  HELP  ", args, ac);
    XtManageChild(help_button);


    XtAddCallback(ok_button, XmNactivateCallback,
  operator_name_cb, (XtPointer)oper_name);

    XtAddCallback(ok_button, XmNactivateCallback, read_timing_cb,
  (XtPointer) op_being_updated);

    XtAddCallback(ok_button, XmNactivateCallback, operator_ok_cb,
  (XtPointer) op_being_updated);

    XtAddCallback(cancel_button, XmNactivateCallback, operator_cancel_cb,
  (XtPointer) op_dialog);

    XtAddCallback(help_button, XmNactivateCallback, operator_help_cb,
  (XtPointer) "operator_property.hlp");


    /*********************************************************/


    XtManageChild(op_dialog);

    XtVaSetValues(op_dialog, XmNx, x, XmNy, y, NULL);

    XtPopup(GetTopShell(op_dialog), XtGrabNone);

    XmProcessTraversal(parent, XmTRAVERSE_CURRENT);
  }

  return;
}
```

```
/* ====================================================================
 *
 *  File: postpopup.h
 *
 *  Description:
 * Header for functions used to post popup menus.
 *
 ==================================================================== */

#ifndef POSTPOUP_H
#define POSTPOUP_H

#include <X11/cursorfont.h>
#include <Xm/RowColumn.h>

#ifdef __cplusplus
extern "C" {
#endif

/* ====================================================================
 *
 *  Function:   PostPopUp
 *
 *  Description:
 *              Event handler to "post" popup menus if the button is
 *              the right mouse button, i.e. third button.
 ==================================================================== */

#ifdef _NO_PROTO

extern void
PostPopUp() ;

#else

void
PostPopUp(
  Widget     w,
  XtPointer  client_data,
  XEvent    *event,
  Boolean   *not_used) ;

#endif /* _NO_PROTO  */

/* ====================================================================
 *
 *  Function:   PostDynamicPopUp
 *
 *  Description:
 *              Event handler to "post" popup dynamic menus if the button is
 *              the right mouse button, i.e. third button.
 ==================================================================== */
#ifdef _NO_PROTO

extern void
PostDynamicPopUp() ;

#else

extern void
PostDynamicPopUp(
  Widget     w,
  XtPointer  client_data,
  XEvent    *event,
  Boolean   *not_used) ;

#endif /* _NO_PROTO */


#ifdef __cplusplus
}  /* Close scope of 'extern "C"' declaration which encloses file. */
#endif

#endif /* ifndef POSTPOUP_H */
```

```
/* ======================================================================
 *
 *  File: postpopup.c
 *
 *  Description:
 * Functions used to post popup menus.
 *
 ====================================================================== */
/*---------------------------------------------------------------------------
    $Source: /home/anunciad/src/master/graphics_editor/postpopup.c,v $
    $Author: anunciad $
    $Date: 1996/08/21 14:20:53 $
---------------------------------------------------------------------------*/
#include <X11/cursorfont.h>
#include <Xm/RowColumn.h>

#include "gettopshell.h"
#include "setcursor.h"

/* ======================================================================
 *
 *  Function:   PostPopUp
 *
 *  Description:
 *              Event handler to "post" popup menus if the button is
 *              the right mouse button, i.e. third button.
 ====================================================================== */
#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

void
PostPopUp(w, client_data, event, not_used)
  Widget     w;
  XtPointer  client_data;
  XEvent    *event;
  Boolean   *not_used;

#else

void
PostPopUp(
  Widget     w,
  XtPointer  client_data,
  XEvent    *event,
  Boolean   *not_used)

#endif /* _NO_PROTO */

{
  Widget Menu = (Widget)client_data;

  if (event->xbutton.button == Button3) {
    XmMenuPosition(Menu, (XButtonPressedEvent *)event);
    XtManageChild(Menu);
  }

  return;
};

/* ======================================================================
 *
 *  Function:   PostDynamicPopUp
 *
 *  Description:
 *              Event handler to "post" popup dynamic menus if the button is
 *              the right mouse button, i.e. third button.
 ====================================================================== */

#ifdef _NO_PROTO

void
PostDynamicPopUp(w, client_data, event, not_used)
  Widget     w;
  XtPointer  client_data;
  XEvent    *event;
  Boolean   *not_used;

#else

void
PostDynamicPopUp(
  Widget     w,
  XtPointer  client_data,
  XEvent    *event,
  Boolean   *not_used)

#endif /* _NO_PROTO */

{
  Widget Menu;
  Widget (*rebuild)() = (Widget (*)())client_data;

  setcursor(GetTopShell(w), True, XC_watch);

  if (event->xbutton.button == Button3) {

    Menu = (*rebuild)();
    XmMenuPosition(Menu, (XButtonPressedEvent *)event);
    XtManageChild(Menu);
  }

  setcursor(GetTopShell(w), False, None);
```

```
  return;
};

#ifdef __cplusplus
```

```
} /* Close scope of 'extern "C"' declaration which encloses file. */
#endif
```

```
#ifndef REPORT_ERRORS_H
#define REPORT_ERRORS_H 1

void report_errors(ERROR_MSGS errors_present, Widget widget,
   ACTION next_action_ptr, BOOLEAN *return_sde_flag,
      char **prev_status);

#endif
```

```
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>


#include <X11/Xatom.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/Form.h>
#include <Xm/LabelG.h>
#include <Xm/List.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectioB.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/ToggleBG.h>
#include <Xm/Xm.h>

#include "ge_interface.h"
#include "graph_editor.h"
#include "warning.h"
#include "action_area.h"

#define OP_LABEL_LEN 15

typedef struct rep_err_widget_data {
  Widget      widget;
  ERROR_MSGS  errors_present;
  ACTION      next_action_ptr;
  BOOLEAN    *return_sde_flag;
  Widget      parent_w;
  char       **prev_status;
} Rep_Err_Data, *Rep_Err_Data_Ptr;


static void errs_close_cb(Widget w, XtPointer client_data,
   XtPointer call_data) {

  Rep_Err_Data_Ptr wd = (Rep_Err_Data_Ptr) client_data;

  XtDestroyWidget(wd->parent_w);

  free(wd);

}

static void errs_parent_cb(Widget w, XtPointer client_data,
   XtPointer call_data) {

  int *selected_error, count;
  int i;
  ERROR_MSGS errPtr, errors_present;
  ACTION     next_action_ptr;
  BOOLEAN    *return_sde_flag;
  Widget     list, parent_w;
  char  **prev_status;

  Rep_Err_Data_Ptr wd = (Rep_Err_Data_Ptr) client_data;
  list            = wd->widget;
  errors_present  = wd->errors_present;
  next_action_ptr = wd->next_action_ptr;
  return_sde_flag = wd->return_sde_flag;
  prev_status     = wd->prev_status;
  parent_w        = wd->parent_w;

  if (XmListGetSelectedPos(list, &selected_error, &count)) {

    i = selected_error[0];
    XtFree((char*) selected_error);
    errPtr = errors_present;
    while (--i)
      errPtr = errPtr->next;

    next_action_ptr->option      = CHECK_SYNTAX;
    next_action_ptr->reinvoke    = true;
    free(next_action_ptr->next_op);
    next_action_ptr->next_op     = dup_str(errPtr->parent_op_label);
    next_action_ptr->next_op_num = errPtr->parent_op_num;
    *return_sde_flag             = true;
    free((char*) *prev_status);
    *prev_status                 = dup_str(errPtr->msg);

    XtDestroyWidget(wd->parent_w);    /* kbm 97/02/26 */
    free(wd);                         /* kbm 97/02/26 */

  }
  else
    warning(parent_w, "Please select an error message frist");
  return;
}

static void errs_current_cb(Widget w, XtPointer client_data,
   XtPointer call_data) {


  int *selected_error, count;
  int i;
  ERROR_MSGS errPtr, errors_present;
  ACTION     next_action_ptr;
  BOOLEAN    *return_sde_flag;
```

```
  Widget     list, parent_w;
  char  **prev_status;

  Rep_Err_Data_Ptr wd = (Rep_Err_Data_Ptr) client_data;
  list            = wd->widget;
  errors_present  = wd->errors_present;
  next_action_ptr = wd->next_action_ptr;
  return_sde_flag = wd->return_sde_flag;
  prev_status     = wd->prev_status;
  parent_w        = wd->parent_w;

  if (XmListGetSelectedPos(list, &selected_error, &count)) {

    i = selected_error[0];
    XtFree((char*)selected_error);
    errPtr = errors_present;
    while (--i)
      errPtr = errPtr->next;

    next_action_ptr->option      = CHECK_SYNTAX;
    next_action_ptr->reinvoke    = true;
    free(next_action_ptr->next_op);
    next_action_ptr->next_op     = dup_str(errPtr->cur_op_label);
    next_action_ptr->next_op_num = errPtr->cur_op_num;
    *return_sde_flag             = true;
    free((char*) *prev_status);
    *prev_status                 = dup_str(errPtr->msg);

    XtDestroyWidget(wd->parent_w);    /* kbm 97/02/26 */
    free(wd);                         /* kbm 97/02/26 */

  }
  else
    warning(parent_w, "Please select an error message frist");
  return;
}

void report_errors(ERROR_MSGS errors_present, Widget widget,
   ACTION next_action_ptr, BOOLEAN *return_sde_flag,
   char **prev_status) {

  if (errors_present == NULL) {
    warning( widget, "No PSDL errors");
    return;
  }

  /* build scroll popup */

  static ActionAreaItem action_items[] = {
    {"Close",        errs_close_cb,   NULL},
    {"Goto Parent",  errs_parent_cb,  NULL},
    {"Goto Current", errs_current_cb, NULL},
    {"Help",         help_cb,         "error.hlp"}
};

Widget      dialog, rc, pane, list, action_a;
int         count = 0, i, j, n=0;
ERROR_MSGS  errPtr;
Arg         args[5];
XmString    *str,string;
char        *errStrBuf;
int         errStrLen, strLen;

// Build up list of errors
errPtr = errors_present;

while (errPtr) {  // Count the number of errors present
  count++;
  errPtr = errPtr->next;
}

// Now build an array of pointers to the error messages (XmStrings)
errPtr = errors_present;

str = (XmString *) XtMalloc (count * sizeof (XmString));
for (i = 0; i < count; i++) {
  errStrLen = 2*OP_LABEL_LEN + strlen(errPtr->msg) + 3;
  errStrBuf = (char *) malloc(errStrLen);

  // Clear area for labels
  for (j = 0; j < (2*OP_LABEL_LEN + 2); j++)
    *(errStrBuf+j) = ' ';
  // Add parent_op_label
  strLen = strlen(errPtr->parent_op_label);
  memcpy(errStrBuf, errPtr->parent_op_label,
 (OP_LABEL_LEN < strLen) ? OP_LABEL_LEN : strLen);

  // Now add current_op_label
  strLen = strlen(errPtr->cur_op_label);
  memcpy((errStrBuf+OP_LABEL_LEN+1), errPtr->cur_op_label,
 (OP_LABEL_LEN < strLen) ? OP_LABEL_LEN : strLen);

  // Finish off with the error message
  *(errStrBuf+2*OP_LABEL_LEN+2) = '\0';
  strcat(errStrBuf, errPtr->msg);

  str[i] = XmStringCreateSimple(errStrBuf);
  free(errStrBuf);

  errPtr = errPtr->next;   // Now process next error message
}

dialog = XtVaCreatePopupShell("dialog", xmDialogShellWidgetClass,
           widget,
           XmNtitle, "Error Messages",
           XmNdeleteResponse, XmDESTROY,
```

```
            NULL);

pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
            XmNsashWidth,   1,
            XmNsashHeight,  1,
            NULL);

rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass,
                      pane, NULL);
string = XmStringCreateSimple("Select Error Message");
XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
            XmNlabelString, string,
            NULL);
XmStringFree(string);
string = XmStringCreateSimple("Parent           Current"
                              "            Error Message");
XtVaCreateManagedWidget("label_2", xmLabelGadgetClass, rc,
            XmNlabelString, string,
            NULL);
XmStringFree(string);

list = XmCreateScrolledList(rc, "PSDL Errors", NULL, 0);
XtVaSetValues(list,
      XmNvisibleItemCount,    10,
      XmNitemCount,           count,
      XmNitems,               str,
      NULL);
XtManageChild(list);

for (i = 0; i < count; i++) {
  XmStringFree(str[i]);
}

XtManageChild(rc);

// Make a client_data structure
Rep_Err_Data_Ptr wd = (Rep_Err_Data_Ptr) malloc(sizeof(Rep_Err_Data));
wd->widget          = list;
wd->errors_present  = errors_present;
wd->next_action_ptr = next_action_ptr;
wd->return_sde_flag = return_sde_flag;
wd->parent_w        = (Widget) dialog;
wd->prev_status     = prev_status;

//Set client data
action_items[0].data = (XtPointer)wd;     // Close
action_items[1].data = (XtPointer)wd;     // Goto Parent
action_items[2].data = (XtPointer)wd;     // Goto Current
action_items[3].data = (XtPointer)wd;     // Help

action_a = CreateActionArea(pane, action_items,
                            XtNumber(action_items));

XtManageChild(pane);
XtPopup(dialog, XtGrabNone);

}
```

```
/* ************************************************************

Name:          resources.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 21 Sep 92
Remarks:   Establishes common defines for the different
         colors and fonts used in the graph_editor.

************************************************************ */
#ifndef RESOURCES_H
#define RESOURCES_H 1

#define MAXCOLORS 64
#define AQUAMARINE 1
#define BLACK 2
#define BLUE 3
#define BLUEVIOLET 4
#define BROWN 5
#define CADETBLUE 6
#define CORAL 7
#define CORNFLOWERBLUE 8
#define CYAN 9
#define DARKGREEN 10
#define DARKOLIVEGREEN 11
#define DARKORCHID 12
#define DARKSLATEBLUE 13
#define DARKSLATEGREY 14
#define DARKTURQUOISE 15
#define DIMGREY 16
#define FIREBRICK 17
#define FORESTGREEN 18
#define GOLD 19
#define GOLDENROD 20
#define GREY 21
#define GREEN 22
#define GREENYELLOW 23
#define INDIANRED 24
#define KHAKI 25
#define LIGHTBLUE 26
#define LIGHTGREY 27
#define LIGHTSTEELBLUE 28
#define LIMEGREEN 29
#define MAGENTA 30
#define MAROON 31
#define MEDIUMAQUAMARINE 32
#define MEDIUMBLUE 33
#define MEDIUMORCHID 34
#define MEDIUMSEAGREEN 35
#define MEDIUMSLATEBLUE 36
#define MEDIUMSPRINGGREEN 37
#define MEDIUMTURQUOISE 38
#define MEDIUMVIOLETRED 39
#define MIDNIGHTBLUE 40
#define NAVY 41
#define ORANGE 42
#define ORANGERED 43
#define ORCHID 44
#define PALEGREEN 45
#define PINK 46
#define PLUM 47
#define RED 48
#define SALMON 49
#define SEAGREEN 50
#define SIENNA 51
#define SKYBLUE 52
#define SLATEBLUE 53
#define SPRINGGREEN 54
#define STEELBLUE 55
#define TAN 56
#define THISTLE 57
#define TURQUIOSE 58
#define VIOLET 59
#define VIOLETRED 60
#define WHEAT 61
#define WHITE 62
#define YELLOW 63
#define YELLOWGREEN 64

#define MAXFONTS 6
#define COURIERBOLD10 1
#define COURIERBOLD12 2
#define COURIERBOLD14 3
#define COURIERMED10 4
#define COURIERMED12 5
#define COURIERMED14 6

#endif
```

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include "inter_process_utilities.h"
#include "sde_simulator.h"

extern int sde_to_ge_channel[];
extern int ge_to_sde_channel[];
extern XferBuffer* xfer;

void write_error_string(s)
  char *s;
{
  printf("error in parent: %s\n", s);
}

main()
{

  int child_process;

  char
ge_in[100],
ge_out[100];

  if (pipe(sde_to_ge_channel) < 0)
  {
    write_error_string("cannot establish sde_to_ge_channel");
    return;
  }

  if (pipe(ge_to_sde_channel) < 0)
  {
    write_error_string("cannot establish ge_to_sde_channel");
    return;
  }

  sprintf(ge_in,  "%d", sde_to_ge_channel[0]);
  sprintf(ge_out, "%d", ge_to_sde_channel[1]);

  if ((child_process = fork()) == -1)
  {
    write_error_string("cannot establish GE process");
    return;
  }

  if (child_process)
  {
      /* this is still the parent */


    int
        SDEchkWord = CHKWORD,
        GEchkWord;


    ACTION_NODE     next_action;
    ACTION          action = &next_action;

    GRAPH_DESC_NODE graph;
    GRAPH_DESC      current_graph = &graph;

    ERROR_MSGS      sde_error_msgs = NULL;

    char kill_buf[50];

    xfer = NULL;
    create_xfer_buf(&xfer);

    init_action(action);
    init_graph_desc(current_graph);
    next_action.reinvoke = true;

    close(sde_to_ge_channel[0]);
    close(ge_to_sde_channel[1]);

    writeChkWord(SDEchkWord, sde_to_ge_channel[1]);
    readChkWord((int *) &GEchkWord, ge_to_sde_channel[0]);

    if (SDEchkWord == GEchkWord) {
      init_sde_sim(current_graph, &sde_error_msgs);
      while (next_action.reinvoke) {

simSDE(xfer, sde_to_ge_channel[1], ge_to_sde_channel[0],
       current_graph, action, sde_error_msgs, child_process);

      }
      printf("SDE exiting...\n");
      printf("\nSDE Xfer Buffer size: %d\n", xfer->Max);

      sprintf(kill_buf, "kill -9 %d ", child_process);
      system(kill_buf);

    }
    else {
      printf("Error opening graphics editor\n");
    }

  }
  else
  {
    char child_buf[2];

    /* this is the child process */
    close(sde_to_ge_channel[1]);
```

```
    close(ge_to_sde_channel[0]);

    execl("edit_graph", "edit_graph -geom 1200x750", ge_in, ge_out, (char *)0);

/*
    close(sde_to_ge_channel[0]);
    close(ge_to_sde_channel[1]);
*/
```

```
        write_error_string("cannot execute graph_editor");
        return;
    }
}
```

```
#ifndef sde_simulator_h
#define sde_simulator_h 1

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

void simSDE();
void init_sde_sim();

#else

void simSDE(XferBuffer *xfer, int ChlWrite, int ChlRead,
    GRAPH_DESC current_graph, ACTION action, ERROR_MSGS SDEerrs);
void init_sde_sim(GRAPH_DESC current_graph, ERROR_MSGS sde_error_msgs);

#endif

#ifdef __cplusplus
}
#endif

#endif
```

```
#include <stdio.h>

#include "ge_utilities.h"
#include "sde_simulator.h"
#include "inter_process_utilities.h"

void init_sde_sim(current_graph, sde_error_msgs)

GRAPH_DESC current_graph;
ERROR_MSGS *sde_error_msgs;

{
  FILE *fp;
  ERROR_MSGS Err;

  init_graph_desc(current_graph);

  /* Initialize added stuff that is not read in from file */
  current_graph->input_list                = NULL;
  current_graph->output_list               = NULL;
  current_graph->timer_list                = NULL;
  current_graph->graph_informal_desc       = NULL;
  current_graph->avail_impl_langs          = NULL;
  current_graph->global_types              = NULL;

  if (fp = fopen("GRAPH_DESC","r")) {
    read_gdn_file(current_graph,fp);
    /*    display_gdn(current_graph);  */
    fclose(fp);
  }
  else {
    current_graph->root_op_name = (char *) dup_str("A");
    current_graph->root_op_num  = 1;
    current_graph->parent_op_name = (char *) dup_str("B");
    current_graph->parent_op_num  = 2;
    current_graph->current_op_name = (char *) dup_str("C");
    current_graph->current_op_num  = 3;
    current_graph->cur_op_spec_met = UNDEFINED_TIME;
    current_graph->cur_op_spec_met_unit = 1;
  }
  current_graph->cur_op_spec               = dup_str("Current op spec");
  current_graph->cur_op_is_terminator      = false;
  current_graph->avail_impl_langs                  =
          (ID_LIST) malloc(sizeof(ID_NODE));
  current_graph->avail_impl_langs->id              =
          (char *) dup_str("Ada");
  current_graph->avail_impl_langs->next            =
          (ID_LIST) malloc(sizeof(ID_NODE));
  current_graph->avail_impl_langs->next->id        =
          (char *) dup_str("PSDL");
  current_graph->avail_impl_langs->next->next      =
          (ID_LIST) malloc(sizeof(ID_NODE));
  current_graph->avail_impl_langs->next->next->id  =
          (char *) dup_str("TAEplus");
  current_graph->avail_impl_langs->next->next->next =
          (ID_LIST) NULL;

  Err = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));
  *sde_error_msgs = Err;

  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("Error at line 100");
  Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

  Err = Err->next;
  Err->parent_op_num   = current_graph->current_op_num;
  Err->parent_op_label = dup_str(current_graph->current_op_name);
  Err->cur_op_num      = current_graph->parent_op_num;
  Err->cur_op_label    = dup_str(current_graph->parent_op_name);
  Err->msg             = dup_str("403");
  Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

  Err = Err->next;
  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("A little bit longer error message "
                                 "goes here ");
  Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

  Err = Err->next;
  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("Error 4");
  Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

  Err = Err->next;
  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("Error 5");
  Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

  Err = Err->next;
  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("Error 6");
```

```
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 7");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 8");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 9");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 10");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 11");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 12");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 13");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 14");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 15");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 16");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 17");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 18");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

Err = Err->next;
Err->parent_op_num   = current_graph->parent_op_num;
Err->parent_op_label = dup_str(current_graph->parent_op_name);
Err->cur_op_num      = current_graph->current_op_num;
Err->cur_op_label    = dup_str(current_graph->current_op_name);
Err->msg             = dup_str("Error 19");
Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));
```

```
  Err = Err->next;
  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("Error 20");
  Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

  Err = Err->next;
  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("Error 21");
  Err->next            = (ERROR_MSGS) malloc(sizeof(ERROR_NODE));

  Err = Err->next;
  Err->parent_op_num   = current_graph->parent_op_num;
  Err->parent_op_label = dup_str(current_graph->parent_op_name);
  Err->cur_op_num      = current_graph->current_op_num;
  Err->cur_op_label    = dup_str(current_graph->current_op_name);
  Err->msg             = dup_str("Error 22...last");

  Err->next            = NULL;
}


void simSDE(xfer, ChlWrite, ChlRead, current_graph, action,
            SDEerrs, child)

XferBuffer *xfer;
int ChlWrite;
int ChlRead;
GRAPH_DESC current_graph;
ACTION     action;
ERROR_MSGS SDEerrs;
int        child;

{

  FILE *fp;
  char buffer[100];

  /* Send data */
  writeGraphDesc(current_graph, xfer, ChlWrite);
  writeErrorMsgs(SDEerrs, xfer, ChlWrite);

  /* Wait for return from edit_graph and read return data */
  readGraphDesc(current_graph, xfer, ChlRead);
  readAction(action, xfer, ChlRead);

  if (action->option == SAVE_TO_DISK) { /* Print what edit_graph   */
    printf("SDE: Save psdl\n");         /* returned upon exit.     */
    fp = fopen("GRAPH_DESC","w");
    write_gdn_file(current_graph,fp);
    fclose(fp);
  }
  else if (action->option == ABANDON) {
    printf("SDE: Abandon psdl\n");
    if (fp = fopen("GRAPH_DESC","r")) {
      read_gdn_file(current_graph,fp);
      fclose(fp);
    }
    else
      printf("SDE:  Error finding file GRAPH_DESC\n");
  }
  else if (action->option == REVERT) {
    printf("SDE: Revert psdl\n");
    if (fp = fopen("GRAPH_DESC","r")) {
      read_gdn_file(current_graph,fp);
      fclose(fp);
    }
    else
      printf("SDE:  Error finding file GRAPH_DESC\n");
  }
  else {
    if (action->reinvoke == true) {
      printf("SDE: Reinvoke with operator %s", action->next_op);
      current_graph->current_op_name = action->next_op;
    }
    else {
      sprintf(buffer, "kill -9 %d\n", child);
      system(buffer);
      writeGraphDesc(current_graph, xfer, ChlWrite);
    }
  }
}
```

```
/* ====================================================================
 *
 *  Function: setcursor
 *
 *  Description: set cursor for current menu
 *
 *  Arguments:          parent -- widget id of parent
 *                      on     -- change cursor, True or False
 *                      cursornumber -- cursor to change to
 *
 *  Returned value: void
 *
 *  Warnings:
 *
 *  Author: Not Roy
 *
 *  Created: 08/07/91
 *==================================================================== */
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <X11/cursorfont.h>
#include <X11/Xlib.h>
#include <Xm/Xm.h>

#ifndef SETCURSOR_H
#define SETCURSOR_H

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

extern void setcursor() ;

#else

extern void
setcursor(Widget parent, int on, int cursornumber) ;

#endif /* _NO_PROTO */

#ifdef __cplusplus
}  /* Close scope of 'extern "C"' declaration which encloses file. */
#endif

#endif /* SETCURSOR_H */
```

```
/* ====================================================================
 *
 *  Function: setcursor
 *
 *  Description: set cursor for current menu
 *
 *  Arguments:          parent -- widget id of parent
 *                      on     -- change cursor, True or False
 *                      cursornumber -- cursor to change to
 *
 *  Returned value: void
 *
 *  Warnings:
 *
 *  Author: Not Roy
 *
 *  Created: 08/07/91
 *==================================================================== */
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <X11/cursorfont.h>
#include <X11/Xlib.h>
#include <Xm/Xm.h>

#include "setcursor.h"

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _NO_PROTO

void
setcursor(parent, on, cursornumber)
  Widget parent;
  int on;
  int cursornumber;

#else

void
setcursor(
  Widget parent,
  int on,
  int cursornumber)

#endif /* _NO_PROTO */
{
  Display *xdisplay;
  Window xwindow;
  XSetWindowAttributes attrs;
  Cursor cursor;


  xwindow = XtWindow(parent);
  xdisplay = XtDisplay(parent);

  if (on)
    cursor = XCreateFontCursor(xdisplay, cursornumber);

  attrs.cursor = on? cursor : None;

  XChangeWindowAttributes(xdisplay, xwindow, CWCursor, &attrs);

  XFlush(xdisplay);

  return;
};

#ifdef __cplusplus
}  /* Close scope of 'extern "C"' declaration which encloses file. */
#endif
```

```
/* ************************************************************

Name:          spline_object.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the SplineObject class.

          The SplineObject is used to create the curved lines
        used in the graph_editor's splines.  It stores a
        linked list of control points which it uses to draw
        itself, point by point.

          In order to correctly terminate in the appropriate
        locations, it adds special control points I call
        'shadow points' at the ends of the lines.  Since
        b-splines don't normally end at the first and last
        control points, the extra shadow points make the
        curve stop in the right place.

History:

  @1   96/10/04  Ken Moeller
       Removed un-needed routines to read and write to disk.

  @2   96/10/06  Ken Moeller
       Created a new routine to write spline data to ge_interface
       structure.

************************************************************* */
#ifndef spline_object_h
#define spline_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <Xm/MessageB.h>
#include "ge_defs.h"
#include "ge_interface.h"

class OperatorObject;
class StreamObject;

// S. Decato 8/1/96
// added the following class forward declaration
// to avoid complaints from the friend definition
class SplineObject;

class _spline_node {
  friend SplineObject;
protected:
  int _x, _y;
  _spline_node *_next_ptr;
public:
  _spline_node() {_next_ptr = NULL;}
  _spline_node(int x, int y) {_x = x; _y = y; _next_ptr = NULL;}
  _spline_node(XYPAIR inpair)
    {_x = inpair.x; _y = inpair.y; _next_ptr = NULL;}
  virtual ~_spline_node() {
#ifdef GE_DEBUG
//    printf("_spline_node Destructor: %d  %d  delete _next_ptr\n", _x, _y);
#endif
    delete _next_ptr;}
};

class SplineObject {
protected:
  static Widget _error_tgt;

  _spline_node *_head_ptr;
  BOOLEAN _shadow_pts_set, _handles_drawn;
  _spline_node *_iter;
  int _handle_selected;

  void draw_arrowhead(GC graphics_context, StreamObject *parent,
                      _spline_node *endpoint);
  int num_points(_spline_node *p1, _spline_node *p2,
                 _spline_node *p3, _spline_node *p4);
public:
  static void set_error_tgt(Widget widget) {_error_tgt = widget;}

  SplineObject();
  virtual ~SplineObject() {
#ifdef GE_DEBUG
//    printf("SplineObject Destructor: delete _head_ptr\n");
#endif
    delete _head_ptr; }

  SPLINE_PTR write_to_sde(EXTERN_STATUS status);                    //@2
  void       build_from_sde(SPLINE_PTR sp);
  CLASS_DEF is_a() {return SPLINEOBJECT;}
  SplineObject& operator=(SplineObject&);
  void set_object_ptrs(OperatorObject *_from_ptr,
                       OperatorObject *_to_ptr);
  void draw(StreamObject* parent, GC graphics_context,
            GC handle_context, DRAW_STYLE style,
            EXTERN_STATUS status);
  XYPAIR first_point();
  XYPAIR last_point();
  void draw_handles(GC draw_context, StreamObject *parent,
                    int x1, int y1);
  void clear();
  void add(int x, int y);
  void reset_iter();
  XYPAIR next_pair();
  BOOLEAN hit(int x, int y);
```

```
BOOLEAN over(int x, int y); // Added 8/26/96, dha
void set_text_location(int &name_x, int &name_y,
                       int &latency_x, int &latency_y);
void set_name_location(int &name_x, int &name_y);
void set_latency_location(int name_x, int name_y,
                          int &latency_x, int &latency_y);
void set_offset_defaults(int *x_pos, int *y_pos);
void set_extern_location(XYPAIR &extern_location,
                         EXTERN_STATUS status);
BOOLEAN hit_handle(int x, int y, EXTERN_STATUS status);
void move_handle(int x, int y);
```

```
  BOOLEAN empty() {return _head_ptr == NULL;}
  void erase_handle(GC graphics_context, StreamObject *parent);
  void reset_handles_drawn() {_handles_drawn = FALSE;}
};

#endif
```

```
/* ************************************************************

Name:          spline_object.C
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 17 Sep 92
Remarks:   Implementation of the SplineObject class.

          The SplineObject is used to create the curved lines
        used in the graph_editor's splines.  It stores a
        linked list of control points which it uses to draw
        itself, point by point.

          In order to correctly terminate in the appropriate
        locations, it adds special control points I call
        'shadow points' at the ends of the lines.  Since
        b-splines don't normally end at the first and last
        control points, the extra shadow points make the
        curve stop in the right place.


Credits:    Portions of code are adapted from the following:

         Barakati, Naba, X Window System Programming, SAMS,
           1991.

         Heller, Dan, Motif Programming Manual, O'Reilly and
           Associates, 1991.

         Johnson, Eric, and Reichard, Kevin, X Window
           Applications Programming, MIS Press, 1989.

         Young, Douglas, Object Oriented Programming With C++
           and OSF/Motif, Prentice-Hall, 1992.

           Formula for b-spline curves comes from:

         Zyda, Michael, Book Number 5, Graphics and Video
           Laboratory, Naval Postgraduate School, 1990.

History:
  @1   96/10/03  Ken Moeller
       Spline loop did not exit.

  @2   96/10/04  Ken Moeller
       Removed code to read and write to disk.

  @3   96/10/06  Ken Moeller
       Created a new routine to write spline data to ge_interface
       structure.

************************************************************ */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stream.h>
#include "ge_defs.h"
#include "spline_object.h"
#include "stream_object.h"
#include "operator_object.h"
#include "warning.h"

#define ARROWANGLE 45.0
#define ARROWSIDELENGTH 10.0
#define HITDISTANCE 10
#define SKIPFACTOR 4
#define NONE 0
#define EXTERN_OFFSET 30

//   Initializes the static error target widget.

Widget SplineObject::_error_tgt = NULL;

SplineObject::SplineObject() {

  _head_ptr        = NULL;
  _iter            = NULL;
  _shadow_pts_set  = FALSE;
  _handle_selected = NONE;
  _handles_drawn   = FALSE;
}


//   Draws a handle at the given coordinates.

void SplineObject::draw_handles(GC draw_context,
                                StreamObject *parent, int x1,
                                int y1) {
#ifdef GEDEBUG
        cout << "Spline: " << (x1 - (HANDLESIZE / 2)) << " " <<
          (y1 - (HANDLESIZE / 2)) << " " << HANDLESIZE << endl;
#endif
        XSetFunction(parent->display_ptr(), draw_context, GXxor);
        XFillRectangle(parent->display_ptr(),
                       parent->draw_window(),
                       draw_context, (x1 - (HANDLESIZE / 2)),
                       (y1 - (HANDLESIZE / 2)), HANDLESIZE,
                       HANDLESIZE);
        XFillRectangle(parent->display_ptr(),
                       *(parent->drawing_area_pixmap()),
                       draw_context, (x1 - (HANDLESIZE / 2)),
                       (y1 - (HANDLESIZE / 2)), HANDLESIZE,
                       HANDLESIZE);
        XSetFunction(parent->display_ptr(), draw_context, GXcopy);
}
```

```
//   Assignment operator.

SplineObject& SplineObject::operator=(SplineObject& spline) {
  _spline_node *source_temp_ptr, *target_temp_ptr;

  if(spline._head_ptr != NULL) {
    _head_ptr = new _spline_node;
    _head_ptr->_x = spline._head_ptr->_x;
    _head_ptr->_y = spline._head_ptr->_y;
    source_temp_ptr = spline._head_ptr->_next_ptr;
    target_temp_ptr = _head_ptr;
    while(source_temp_ptr != NULL) {
      target_temp_ptr->_next_ptr = new _spline_node;
      target_temp_ptr = target_temp_ptr->_next_ptr;
      target_temp_ptr->_x = source_temp_ptr->_x;
      target_temp_ptr->_y = source_temp_ptr->_y;
      source_temp_ptr = source_temp_ptr->_next_ptr;
    }
  }
  return *this;
}

//   Returns the coordinates of the first point in the spline.

XYPAIR SplineObject::first_point() {
  XYPAIR temp_pair;

  if(_head_ptr != NULL) {
    temp_pair.x = _head_ptr->_x;
    temp_pair.y = _head_ptr->_y;
  }
  else {
    temp_pair.x = 0;
    temp_pair.y = 0;
  }
  return temp_pair;
}

//   Returns the coordinates of the last point in the spline.

XYPAIR SplineObject::last_point() {
  XYPAIR temp_pair;
  _spline_node *temp_node_ptr = _head_ptr;

  if(_head_ptr != NULL) {
    while(temp_node_ptr->_next_ptr != NULL)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_pair.x = temp_node_ptr->_x;
    temp_pair.y = temp_node_ptr->_y;
  }
  else {
    temp_pair.x = 0;
    temp_pair.y = 0;
  }
  return temp_pair;
}

//   Sets _from_ptr and _to_ptr to the operators at either
// end of the spline.  Pointers for external streams equal
// NULL.  Also adds the necessary shadow and intercept points,
// and adds a control point for streams defined without one.
// Splines with externals must always have at least one point.

//   Streams with external endpoints use the last control
// point as the physical endpoint of the spline. Otherwise,
// the attached operator provides the coordinates for the
// intercept point, i.e. the spline endpoint.

//   In a spline with shadow points, the points are stored
// in the linked list in the following order:
//
//        'to' shadow point - 'to' intercept point -
//                (defined control points) -
//        'from' intercept point - 'from' shadow point

void SplineObject::set_object_ptrs(OperatorObject *_from_ptr,
                                   OperatorObject *_to_ptr) {
  XYPAIR from_intercept, to_intercept, temp_pair, first_point,
    last_point;
  _spline_node *temp_node_ptr, *temp_head_ptr, *temp_node_ptr2;

  if(_shadow_pts_set == TRUE) {

//   strips off existing intercept and shadow points. For
// externals, the intercept points are saved.

    temp_node_ptr = _head_ptr;
    while(temp_node_ptr->_next_ptr->_next_ptr->_next_ptr != NULL)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    if(_to_ptr == NULL) {
      temp_pair.x = temp_node_ptr->_next_ptr->_x;
      temp_pair.y = temp_node_ptr->_next_ptr->_y;
      to_intercept = temp_pair;
    }
    delete temp_node_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = NULL;
    temp_node_ptr2 = _head_ptr;
    temp_node_ptr = _head_ptr->_next_ptr->_next_ptr;
    temp_node_ptr2->_next_ptr->_next_ptr = NULL;
    if(_from_ptr == NULL) {
      temp_pair.x = _head_ptr->_next_ptr->_x;
      temp_pair.y = _head_ptr->_next_ptr->_y;
      from_intercept = temp_pair;
    }
```

```
    delete temp_node_ptr2;
    _head_ptr = temp_node_ptr;
  }
  else {
    _shadow_pts_set = TRUE;
    if(_from_ptr == NULL) {
      if(_head_ptr == NULL) {
        warning(_error_tgt,
"External streams must have at least one control point");
        return;
      }
      else {
        temp_node_ptr = _head_ptr;
        _head_ptr = _head_ptr->_next_ptr;
        from_intercept.x = temp_node_ptr->_x;
        from_intercept.y = temp_node_ptr->_y;
        temp_node_ptr->_next_ptr = NULL;
        delete temp_node_ptr;
      }
    }
    if(_to_ptr == NULL) {
      if(_head_ptr == NULL)
        warning(_error_tgt,
"External streams must have at least one control point");
      else {
        temp_node_ptr = _head_ptr;
        if(temp_node_ptr->_next_ptr == NULL) {
          to_intercept.x = temp_node_ptr->_x;
          to_intercept.y = temp_node_ptr->_y;
          delete temp_node_ptr;
          _head_ptr = NULL;
        }
        else {
          while(temp_node_ptr->_next_ptr->_next_ptr != NULL)
            temp_node_ptr = temp_node_ptr->_next_ptr;
          to_intercept.x = temp_node_ptr->_next_ptr->_x;
          to_intercept.y = temp_node_ptr->_next_ptr->_y;
          delete temp_node_ptr->_next_ptr;
          temp_node_ptr->_next_ptr = NULL;
        }
      }
    }
  }
  if(_head_ptr != NULL) {
    temp_pair = this->first_point();
    if(_from_ptr != NULL)
      from_intercept = _from_ptr->intercept(temp_pair.x,
                                            temp_pair.y);
    first_point.x = from_intercept.x -
      (temp_pair.x - from_intercept.x);
    first_point.y = from_intercept.y -
      (temp_pair.y - from_intercept.y);
    temp_pair = this->last_point();
    if(_to_ptr != NULL)
      to_intercept = _to_ptr->intercept(temp_pair.x,
                                        temp_pair.y);
    last_point.x = to_intercept.x - (temp_pair.x -
                                     to_intercept.x);
    last_point.y = to_intercept.y - (temp_pair.y -
                                     to_intercept.y);
    temp_head_ptr = new _spline_node(first_point);
    temp_head_ptr->_next_ptr = new _spline_node(from_intercept);
    temp_node_ptr = temp_head_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = _head_ptr;
    while(temp_node_ptr->_next_ptr != NULL)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = new _spline_node(to_intercept);
    temp_node_ptr->_next_ptr->_next_ptr =
      new _spline_node(last_point);
    _head_ptr = temp_head_ptr;
  }
  else {
    if(_to_ptr == NULL)
      from_intercept = _from_ptr->intercept(to_intercept.x,
                                            to_intercept.y);
    else
      if(_from_ptr == NULL)
        to_intercept =
          _to_ptr->intercept(from_intercept.x,
                             from_intercept.y);
      else {
        temp_pair = _to_ptr->center();
        from_intercept = _from_ptr->intercept(temp_pair.x,
                                              temp_pair.y);
        to_intercept = _to_ptr->intercept(from_intercept.x,
                                          from_intercept.y);
      }
    temp_pair.x = (from_intercept.x + to_intercept.x) / 2;
    temp_pair.y = (from_intercept.y + to_intercept.y) / 2;
    first_point.x = from_intercept.x - (temp_pair.x -
                                        from_intercept.x);
    first_point.y = from_intercept.y - (temp_pair.y -
                                        from_intercept.y);
    last_point.x = to_intercept.x - (temp_pair.x -
                                     to_intercept.x);
    last_point.y = to_intercept.y - (temp_pair.y -
                                     to_intercept.y);
    temp_head_ptr = new _spline_node(first_point);
    temp_head_ptr->_next_ptr = new _spline_node(from_intercept);
    temp_node_ptr = temp_head_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = new _spline_node(temp_pair);
    temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = new _spline_node(to_intercept);
    temp_node_ptr->_next_ptr->_next_ptr =
      new _spline_node(last_point);
    _head_ptr = temp_head_ptr;
```

```
  }
}

//   Determines the number of points to be calculated for the
// spline.

int SplineObject::num_points(_spline_node *p1, _spline_node *p2,
                             _spline_node *p3,
                             _spline_node *p4) {
  int distance, number_points = 0;

  if(p4 != NULL) {
    distance = abs(p1->_x - p2->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_x - p3->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_x - p4->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_x - p3->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_x - p4->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p3->_x - p4->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_y - p2->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_y - p3->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_y - p4->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_y - p3->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_y - p4->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p3->_y - p4->_y);
    if(distance > number_points)
      number_points = distance;
  }
  return number_points;
}

//   Draws the arrowheads for the line.

void SplineObject::draw_arrowhead(GC graphics_context,
                                  StreamObject *parent,
                                  _spline_node *endpoint) {
  _spline_node *last_point = endpoint->_next_ptr;
  double angle, temp_angle;
  double half_arrow_angle = ARROWANGLE / 2.0 * M_PI / 180.0;
  XPoint vertices[3];

  if(last_point->_x == endpoint->_x) {
    if(last_point->_y > endpoint->_y)
      angle = M_PI / 2.0;
    else
      angle = 3.0 * M_PI / 2.0;
  }
  else {
    angle = atan((double) (last_point->_y - endpoint->_y) /
                 (double) (last_point->_x - endpoint->_x));
    if((last_point->_x < endpoint->_x))
      angle = M_PI + angle;
  }
  vertices[0].x = endpoint->_x;
  vertices[0].y = endpoint->_y;
  temp_angle = angle - half_arrow_angle;
  vertices[1].x = (short) (endpoint->_x - (cos(temp_angle)
                                           * ARROWSIDELENGTH));
  vertices[1].y = (short) (endpoint->_y - (sin(temp_angle) *
                                           ARROWSIDELENGTH));
  temp_angle = angle + half_arrow_angle;
  vertices[2].x = (short) (endpoint->_x - (cos(temp_angle) *
                                           ARROWSIDELENGTH));
  vertices[2].y = (short) (endpoint->_y - (sin(temp_angle) *
                                           ARROWSIDELENGTH));
  XFillPolygon(parent->display_ptr(), parent->draw_window(),
               graphics_context, vertices, 3, Convex,
               CoordModeOrigin);
  XFillPolygon(parent->display_ptr(),
               *(parent->drawing_area_pixmap()),
               graphics_context, vertices, 3, Convex,
               CoordModeOrigin);
}

//   Draws the spline point by point, and if the coordinates
// are near a spline point, TRUE is returned.

BOOLEAN SplineObject::hit(int in_x, int in_y) {
  int points;
  float inc;
  float t, t2, t3, term1, term2, term3, term4, x, y;
  _spline_node *p1, *p2, *p3, *p4;

  p1 = _head_ptr;
  p2 = p1->_next_ptr;
```

```
  p3 = p2->_next_ptr;
  p4 = p3->_next_ptr;
  points = num_points(p1, p2, p3, p4) / SKIPFACTOR;
  while(p4 != NULL) {
    inc = 1.0 / (float) points;
    for(t = 0.0; t <= 1.0; t += inc) {
      t2 = t * t;
      t3 = t2 * t;
      term1 = (-t3 + (3 * t2) - (3 * t) +1);
      term2 = ((3 * t3) - (t2 * 6) + 4);
      term3 = ((-3 * t3) + (3 * t2) + (3 * t) + 1);
      term4 = t3;
      x =  term1 * p1->_x;
      x += term2 * p2->_x;
      x += term3 * p3->_x;
      x += term4 * p4->_x;
      x /= 6.0;

      y =  term1 * p1->_y;
      y += term2 * p2->_y;
      y += term3 * p3->_y;
      y += term4 * p4->_y;
      y /= 6.0;
      if((abs((int) x - in_x) < HITDISTANCE) &&
         (abs((int) y - in_y) < HITDISTANCE))
        return TRUE;
    }
    p1 = p2;
    p2 = p3;
    p3 = p4;
    p4 = p4->_next_ptr;
    points = num_points(p1, p2, p3, p4) / SKIPFACTOR;
  }
  return FALSE;
}

//   Draws the spline using the control points.

void SplineObject::draw(StreamObject *parent,
                        GC graphics_context, GC handle_context,
                        DRAW_STYLE style,
                        EXTERN_STATUS status) {
  int points, i, j;
  float inc, t, t2, t3, term1, term2, term3, term4, x, y;
  _spline_node *p1, *p2, *p3, *p4, *temp_node_ptr;
  BOOLEAN need_handles;

  p1 = _head_ptr;
  p2 = p1->_next_ptr;
  p3 = p2->_next_ptr;
  p4 = p3->_next_ptr;
  points = num_points(p1, p2, p3, p4);
  while(p4 != NULL) {
```

```
/* Debug  code
    if((p1->_x > 1000) || (p1->_x < 0))
      printf("Bogus draw p1x: %d id: %d", p1->_x, parent->id());
    if((p1->_y > 1000) || (p1->_y < 0))
      printf("Bogus draw p1y: %d id: %d", p1->_y, parent->id());
    if((p2->_x > 1000) || (p2->_x < 0))
      printf("Bogus draw p2x: %d id: %d", p2->_x, parent->id());
    if((p2->_y > 1000) || (p2->_y < 0))
      printf("Bogus draw p2y: %d id: %d", p2->_y, parent->id());
    if((p3->_x > 1000) || (p3->_x < 0))
      printf("Bogus draw p3x: %d id: %d", p3->_x, parent->id());
    if((p3->_y > 1000) || (p3->_y < 0))
      printf("Bogus draw p3y: %d id: %d", p3->_y, parent->id());
    if((p4->_x > 1000) || (p4->_x < 0))
      printf("Bogus draw p4x: %d id: %d", p4->_x, parent->id());
    if((p4->_y > 1000) || (p4->_y < 0))
      printf("Bogus draw p4y: %d id: %d", p4->_y, parent->id());
*/
    inc = 1.0 / (float) points;
    for(t = 0.0; t <= 1.0; t += inc) {
      t2 = t * t;
      t3 = t2 * t;
      term1 = (-t3 + (3 * t2) - (3 * t) +1);
      term2 = ((3 * t3) - (t2 * 6) + 4);
      term3 = ((-3 * t3) + (3 * t2) + (3 * t) + 1);
      term4 = t3;
      x =  term1 * p1->_x;
      x += term2 * p2->_x;
      x += term3 * p3->_x;
      x += term4 * p4->_x;
      x /= 6.0;

      y =  term1 * p1->_y;
      y += term2 * p2->_y;
      y += term3 * p3->_y;
      y += term4 * p4->_y;
      y /= 6.0;
      if(parent->is_state_variable()) { // draw thicker line
        for(i = (int) x - 1; i < (int) x + 1; i++)
          for(j = (int) y - 1; j < (int) y + 1; j++) {
            XDrawPoint(parent->display_ptr(),
                       parent->draw_window(),
                       graphics_context, i, j);
            XDrawPoint(parent->display_ptr(),
                       *(parent->drawing_area_pixmap()),
                       graphics_context, i, j);
          }
      }
      else {
        XDrawPoint(parent->display_ptr(), parent->draw_window(),
                   graphics_context, (int) x, (int) y);
        XDrawPoint(parent->display_ptr(),
                   *(parent->drawing_area_pixmap()),
```

```
                    graphics_context, (int) x, (int) y);
      }
    }
    p1 = p2;
    p2 = p3;
    p3 = p4;
    p4 = p4->_next_ptr;
    points = num_points(p1, p2, p3, p4);
  }
  temp_node_ptr = _head_ptr->_next_ptr;
  if(((_handles_drawn == FALSE) && (style == SOLID)) ||
      ((_handles_drawn == TRUE) && (style == ERASE)))
    need_handles = TRUE;
  else
    need_handles = FALSE;

  if(parent->is_selected() && (status == FROM_EXTERNAL) &&
     need_handles)
      draw_handles(handle_context, parent, temp_node_ptr->_x,
                   temp_node_ptr->_y);
  temp_node_ptr = temp_node_ptr->_next_ptr;
  while(temp_node_ptr->_next_ptr->_next_ptr != NULL) {
    if(parent->is_selected() && need_handles)
      draw_handles(handle_context, parent, temp_node_ptr->_x,
                    temp_node_ptr->_y);
    temp_node_ptr = temp_node_ptr->_next_ptr;
  }
  draw_arrowhead(graphics_context, parent, temp_node_ptr);
  if(parent->is_selected() && (status == TO_EXTERNAL) &&
     need_handles)
    draw_handles(handle_context, parent, temp_node_ptr->_x,
                 temp_node_ptr->_y);
  if(need_handles) {
    if(_handles_drawn == TRUE)
      _handles_drawn = FALSE;
    else
      _handles_drawn = TRUE;
  }
}

//   Erases the control points.

void SplineObject::clear() {

  delete _head_ptr;
  _head_ptr = NULL;
}

//   Adds a new control point to the spline.

void SplineObject::add(int x, int y) {
  _spline_node *temp_node_ptr;

/* Debug code
  if((x > 900) || (x < 0))
    printf("Bogus spline x: %d", x);
  if((y > 900) || (y < 0))
    printf("Bogus spline y: %d", x);
*/
  if(_head_ptr == NULL)
    _head_ptr = new _spline_node(x, y);
  else {
    temp_node_ptr = _head_ptr;
    while(temp_node_ptr->_next_ptr != NULL)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = new _spline_node(x, y);
  }
}

//   Returns the control point pointed to by the iterator
// pointer, then advances the iterator pointer.

XYPAIR SplineObject::next_pair() {
  XYPAIR temp_pair;

  if(_iter == NULL) {
    temp_pair.x = -1;
    temp_pair.y = -1;
  }
  else {
    temp_pair.x = _iter->_x;
    temp_pair.y = _iter->_y;
    _iter = _iter->_next_ptr;
  }
  return temp_pair;
}

//   Resets the iterator pointer to the beginning of the spline.

void SplineObject::reset_iter() {

  _iter = _head_ptr;
}

//   Places the name between the middle two control points.

void SplineObject::set_name_location(int &name_x, int &name_y) {

  _spline_node *temp_node_ptr = _head_ptr;
  int temp_x, temp_y, i, num_nodes = 0;

  while(temp_node_ptr != NULL) {
    num_nodes++;
    temp_node_ptr = temp_node_ptr->_next_ptr;
  }
```

```
  temp_node_ptr = _head_ptr;
  if(_head_ptr != NULL) {
    for(i = 1; i < num_nodes / 2; i++)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_x =
      (temp_node_ptr->_x + temp_node_ptr->_next_ptr->_x) / 2;
    temp_y =
      (temp_node_ptr->_y + temp_node_ptr->_next_ptr->_y) / 2;

    name_x = temp_x;
    name_y = temp_y;
  }
}

//   Places the latency location underneath the name.

void SplineObject::set_latency_location(int temp_x, int temp_y,
                                        int &latency_x,
                                        int &latency_y) {

  latency_x = temp_x;
  latency_y = temp_y + 15;
}

//   Convenience function for setting the text location.

void SplineObject::set_text_location(int &name_x, int &name_y,
                                     int &latency_x,
                                     int &latency_y) {

  set_name_location(name_x, name_y);
  set_latency_location(name_x, name_y, latency_x, latency_y);
}

void SplineObject::set_offset_defaults(int *x_pos, int *y_pos) {

  _spline_node *temp_node_ptr = _head_ptr;
  int temp_x = 0, temp_y = 0, i = 0, num_nodes = 0;

  while(temp_node_ptr != NULL) {
    num_nodes++;
    temp_node_ptr = temp_node_ptr->_next_ptr;
  }

  temp_node_ptr = _head_ptr;
  if (_head_ptr != NULL) {
    for (i = 1; i < num_nodes / 2; i++)
      temp_node_ptr = temp_node_ptr->_next_ptr;

    if (num_nodes%2) {              // odd numer of nodes
      temp_x = temp_node_ptr->_next_ptr->_x;
      temp_y = temp_node_ptr->_next_ptr->_y;
    }
    else {                          // even, take midpoint
      temp_x = (temp_node_ptr->_x + temp_node_ptr->_next_ptr->_x) / 2;
      temp_y = (temp_node_ptr->_y + temp_node_ptr->_next_ptr->_y) / 2;
    }
  }

  *x_pos = temp_x;
  *y_pos = temp_y;

}


//   Checks to see if the coordinates are within any of the
// handles, or within the ends of external streams.

BOOLEAN SplineObject::hit_handle(int x, int y,
                                 EXTERN_STATUS status) {
  int num_handle = 2;
  _spline_node *temp_node_ptr = _head_ptr->_next_ptr;

  if(status != FROM_EXTERNAL) {
    temp_node_ptr = temp_node_ptr->_next_ptr;
    num_handle++;
  }
  while(temp_node_ptr->_next_ptr->_next_ptr != NULL) {
    if((((temp_node_ptr->_x) - (HANDLESIZE / 2) - HITFUDGE)
                                                    <= x) &&
       (x <=
   ((temp_node_ptr->_x) + (HANDLESIZE / 2) + HITFUDGE)) &&
   (((temp_node_ptr->_y) - (HANDLESIZE / 2) - HITFUDGE) <= y) &&
   (y <= ((temp_node_ptr->_y) + (HANDLESIZE / 2) + HITFUDGE))) {
         _handle_selected = num_handle;
         return TRUE;
    }
    temp_node_ptr = temp_node_ptr->_next_ptr;
    num_handle++;
  }
  if(status == TO_EXTERNAL) {
    if((((temp_node_ptr->_x) - (HANDLESIZE / 2) - HITFUDGE)
                                                    <= x) &&
       (x <=
          ((temp_node_ptr->_x) + (HANDLESIZE / 2) + HITFUDGE)) &&
           (((temp_node_ptr->_y) - (HANDLESIZE / 2) - HITFUDGE)
            <= y) &&
          (y <= ((temp_node_ptr->_y) + (HANDLESIZE / 2) +
                 HITFUDGE))) {
         _handle_selected = num_handle;
         return TRUE;
    }
  }
  _handle_selected = NONE;
  return FALSE;
}
```

```
//   Moves a spline handle.

void SplineObject::move_handle(int x, int y) {
  _spline_node *temp_node_ptr = _head_ptr;
  int i;

  for(i = 1; i < _handle_selected; i++)
    temp_node_ptr = temp_node_ptr->_next_ptr;
  temp_node_ptr->_x += x;
  temp_node_ptr->_y += y;
  if(temp_node_ptr->_x < 0)
    temp_node_ptr->_x = 0;
  if(temp_node_ptr->_y < 0)
    temp_node_ptr->_y = 0;
}

//   Erases the given handle

void SplineObject::erase_handle(GC graphics_context,
                                StreamObject *parent) {
  _spline_node *temp_node_ptr = _head_ptr;
  int i;

  for(i = 1; i < _handle_selected; i++)
    temp_node_ptr = temp_node_ptr->_next_ptr;
  draw_handles(graphics_context, parent, temp_node_ptr->_x,
               temp_node_ptr->_y);
}

//   Determines the location for the 'EXTERNAL' label on
// the drawing.

void SplineObject::set_extern_location(XYPAIR &extern_location,
                       EXTERN_STATUS status) {
  _spline_node *temp_node_ptr, *intercept_ptr, *shadow_ptr;
  int temp_x, temp_y;

  if(status == FROM_EXTERNAL) {
    shadow_ptr = _head_ptr;
    intercept_ptr = _head_ptr->_next_ptr;
  }
  else
    if(status == TO_EXTERNAL) {
      temp_node_ptr = _head_ptr;
      while(temp_node_ptr->_next_ptr->_next_ptr != NULL)
        temp_node_ptr = temp_node_ptr->_next_ptr;
      intercept_ptr = temp_node_ptr;
      shadow_ptr = temp_node_ptr->_next_ptr;
    }
  if(shadow_ptr->_y > intercept_ptr->_y)
    temp_y = intercept_ptr->_y + 7;
  else
    temp_y = intercept_ptr->_y - 7;
  if(shadow_ptr->_y == intercept_ptr->_y)
    if(shadow_ptr->_x > intercept_ptr->_x)
      temp_x = EXTERN_OFFSET;
    else
      temp_x = -EXTERN_OFFSET;
  else
    temp_x = (temp_y - intercept_ptr->_y) /
      ((shadow_ptr->_y) - (intercept_ptr->_y)) *
        ((shadow_ptr->_x) - (intercept_ptr->_x));
  if(abs(temp_x) > EXTERN_OFFSET)
    if(shadow_ptr->_x > intercept_ptr->_x)
      temp_x = EXTERN_OFFSET;
    else
      temp_x = -EXTERN_OFFSET;
  temp_x += intercept_ptr->_x;
  temp_y += 10;
  extern_location.x = temp_x;
  extern_location.y = temp_y;
}

SPLINE_PTR SplineObject::write_to_sde(EXTERN_STATUS status) {  // @3

  SPLINE_PTR    head  = NULL;
  SPLINE_PTR    spPtr = NULL;
  _spline_node *snPtr = _head_ptr;

  if (status == FROM_EXTERNAL)
    snPtr = snPtr->_next_ptr;
  else
    snPtr = snPtr->_next_ptr->_next_ptr;
  while (snPtr->_next_ptr->_next_ptr != NULL) {
    if (spPtr == NULL) {
      head  = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
      spPtr = head;
    }
    else {
      spPtr->next = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
      spPtr = spPtr->next;
    }

    spPtr->x    = snPtr->_x;
    spPtr->y    = snPtr->_y;
    spPtr->next = NULL;
    snPtr = snPtr->_next_ptr;
  }
  if (status == TO_EXTERNAL) {
    spPtr->next = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
    spPtr = spPtr->next;

    spPtr->x    = snPtr->_x;
    spPtr->y    = snPtr->_y;
    spPtr->next = NULL;
```

```
  }
  return head;
}


void SplineObject::build_from_sde(SPLINE_PTR sp) {

  _spline_node *temp_node_ptr;

  this->clear();

  _iter           = NULL;
  _shadow_pts_set  = FALSE;
  _handle_selected = NONE;
  _handles_drawn   = FALSE;

  temp_node_ptr = NULL;

  while (sp != NULL) {
    if (_head_ptr == NULL) {
      _head_ptr = new _spline_node(sp->x, sp->y);
      temp_node_ptr = _head_ptr;
    }
    else {
      temp_node_ptr->_next_ptr = new _spline_node(sp->x, sp->y);
      temp_node_ptr = temp_node_ptr->_next_ptr;
    }
    sp = sp->next;
  }
}
```

```
/* ************************************************************

Name:          stream.h
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the StreamObject class.

           The StreamObject is a graphical representation of
         a curved PSDL stream.  The stream is drawn as a
         b-spline specified by a series of control points.
         Streams automatically connect to their attached
         operators, and always have arrowheads at their
         terminating points.

Reengineering:
Added constructor to support SDE interface 9/10/96

History:

  @1   96/10/04  Ken Moeller
       Removed un-needed routines to read and write to disk.

  @2   96/10/06  Ken Moeller
       Removed un-needed routines to read and write to property.

  @3   96/10/06  Ken Moeller
       Fixes to time units.

  @4   96/10/06  Ken Moeller
       New build_st function.

  @5   96/10/06  Ken Moeller
       Added extra data item access members for new data items.


************************************************************ */
#ifndef stream_object_h
#define stream_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include "ge_defs.h"
#include "ge_interface.h" //Added by DL 9/11/96
#include "graph_object.h"
#include "spline_object.h"
#include "operator_object.h"

class GraphObjectList;

class StreamObject : public GraphObject {
protected:
  char    *_latency_string_ptr;
  int     _name_height, _name_width;
  int     _latency_width, _latency_height;
  int     _handle_selected;
  BOOLEAN _is_selected,
          _name_selected,
          _latency_selected,
          _st_handles_drawn,
          _name_handles_drawn,
          _latency_handles_drawn;
  XYPAIR  name_location,
          lat_location,
          extern_location;

  int     _x,       // Location from which name and latency are offset
          _y;

//--------------------  ge_interface.h  ------------------------

  ST_ID _id;

  char *_name_ptr;
  int
        _name_font,
        _name_x,
        _name_y;

  OP_ID _from,
        _to;

  OperatorObject
        *_from_ptr,
        *_to_ptr;

  SplineObject
        _arc;

  int   _latency,
        _latency_unit,
        _latency_font,
        _latency_x,
        _latency_y;

  char* _stream_type_name;

  char* _state_initial_value;

  BOOLEAN
        _is_state_variable;

  BOOLEAN
        _is_new,
        _is_modified,
        _is_deleted;
```

```
//-----------------------------------------------------------------


  void set_default_text_location();
  void StreamObject::draw_handles(GC draw_context, int x1,
                                  int y1, int x2, int y2);

public:

  StreamObject();
  StreamObject(STREAM st);
  StreamObject(char *in_name_ptr, OP_ID in_id, OP_ID  in_from,
               OP_ID in_to, int in_latency, int in_unit,          //@3
       SplineObject *in_arc_ptr,
               BOOLEAN in_is_new, BOOLEAN in_is_state_variable);
  virtual ~StreamObject();

  StreamObject& operator=(StreamObject& src);
  void copy(STREAM &st, OP_LIST op_list);
  void read_from(STREAM st);
  void write_to(STREAM StPtr, OP_LIST op_list);
  STREAM build_st(StreamObject *st, OP_LIST op_list);                 // @4
  STREAM copy(OP_LIST op_list);
  STREAM clone(OP_LIST op_list);
  void copy(StreamObject *src);
  void release();
  void initialize();

  CLASS_DEF is_a() {return STREAMOBJECT;}

  void draw(DRAW_STYLE style);
  void erase();
  void erase_handle() {_arc.erase_handle(_graphics_context, this);}
  void draw_spline(DRAW_STYLE style);
  void draw_text(DRAW_STYLE style);
  void erase_text();
  void move_text(int x, int y);

  void select();
  void unselect();

  BOOLEAN hit(int x, int y);
  BOOLEAN hit_handle(int x, int y);
  BOOLEAN over(int x, int y); // Added 8/26/96, dha

  BOOLEAN is_selected() {return _is_selected;}
  BOOLEAN text_selected()
               {return (_name_selected || _latency_selected);}

  BOOLEAN spline_empty() {return _arc.empty();}

  int text_height();
  int text_width();

  void delete_notify(CLASS_DEF class_type, OP_ID deleted_obj_id);
  void undelete_notify(CLASS_DEF class_type, OP_ID deleted_obj_id);
  void move_notify(CLASS_DEF object_type, OP_ID object_id);
  void move_handle(int x, int y) {_arc.move_handle(x, y);}
  void reset_handles_drawn_state();
  void set_object_font(int font_id);
  void set_object_ptrs(GraphObjectList *parent);
  void set_text_dimensions();
  void text_locate(int x, int y);
  void set_default_name_location();
  void set_default_latency_location();

  void inherit_type(StreamObject *fromPtr);

  SPLINE_PTR write_spline_to_sde(EXTERN_STATUS status)
    { return _arc.write_to_sde(status); }

  int x()                    {return _x;}
  int y()                    {return _y;}
  void x(int x)              {_x = x;}
  void y(int y)              {_y = y;}

//----------------------  Methods to Get values  ----------------  //@5
// StreamObject

  ST_ID  id()                {return _id;}

  char *name()               {return dup_str(_name_ptr);}
  int  name_font()           {return _name_font;}
  int  name_x()              {return _name_x;}
  int  name_y()              {return _name_y;}

  OP_ID  from()              {return _from;}
  OP_ID  to()                {return _to;}

  OperatorObject* from_ptr()  {return _from_ptr;}
  OperatorObject* to_ptr()    {return _to_ptr;}

  SplineObject arc()         {return _arc;}

  int  latency()             { return _latency; }
  int  latency_unit()        { return _latency_unit; }
  int  latency_font()        { return _latency_font; }
  int  latency_x()           { return _latency_x; }
  int  latency_y()           { return _latency_y; }

  char* stream_type_name()    {return dup_str(_stream_type_name);}

  char* state_initial_value() {return dup_str(_state_initial_value);}

  BOOLEAN is_state_variable() {return _is_state_variable;}
```

```
  BOOLEAN is_new()            {return _is_new;}
  BOOLEAN is_modified()       {return _is_modified;}
  BOOLEAN is_deleted()        {return _is_deleted;}

//----------------------  Methods to Set values  ----------------

  void id(ST_ID id)             {_id = id;};

  void name(char *new_name);
  void replace_name(char *new_name);
  void name_font(int name_font) {_name_font = name_font;}
  void name_x(int x)            {_name_x = x;}
  void name_y(int y)            {_name_y = y;}

  void from(OP_ID f)               {_from = f;     _from_ptr = NULL;}
  void to(OP_ID t)                 {_to = t;       _to_ptr   = NULL;}
  void from_ptr(OperatorObject *O) {_from_ptr = O; _from = O->id();}
  void to_ptr(OperatorObject *O)   {_to_ptr = O;   _to   = O->id();}

  void arc(SplineObject arc_ptr) {_arc = arc_ptr;}

  void latency(int t)           { _latency = t; }
  void latency_unit(int units)  { _latency_unit = units; }
  void latency(int latency, int unit);
  void latency_font(int font)   { _latency_unit = font; }

  void latency_x(int x)         { _latency_x = x; }

  void latency_y(int y)         { _latency_y = y; }

  void stream_type_name(char *in_ptr) {
            delete _stream_type_name;
     _stream_type_name = dup_str(in_ptr);
   }

  void state_initial_value(char* in_ptr) {
            delete _state_initial_value;
     _state_initial_value = dup_str(in_ptr);
   }

  void is_state_variable(BOOLEAN state)
           {_is_state_variable = state;}

  void set_modified() {_is_modified = true;}
  void set_deleted() {_is_deleted = true;}

//
//-------------------------------------------------------------   //@5

};

#endif;
```

```
/* ************************************************************

Name:          stream_object.C
Author:        Capt Robert M. Dixon
Program:       graph_editor
Date Modified: 11 Sep 92
Remarks:   Implementation of the StreamObject class.

           The StreamObject is a graphical representation of
         a curved PSDL stream.  The stream is drawn as a
         b-spline specified by a series of control points.
         Streams automatically connect to their attached
         operators, and always have arrowheads at their
         terminating points.


Credits:    Portions of code are adapted from the following:

          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.

History:
  @1   96/10/03  Ken Moeller
       Latency did not recognized UNDEFINED_TIME.
       PROBLEM TO FIX:  _latency is still using negative values.

  @2   96/10/04  Ken Moeller
       Removed code to read and write to disk.

  @3   96/10/06  Ken Moeller
       Fixes to latency units.

  @4   96/10/06  Ken Moeller
       Removed build_from_property and write_to_property.

  @5   96/10/06  Ken Moeller
       New build_st function.

************************************************************ */
#include <stdlib.h>
#include <string.h>
#include <stream.h>
#include "stream_object.h"
#include "graph_object_list.h"
#include "ge_interface.h"              // @1

#define MAX_CONTROL_POINTS 100

StreamObject::StreamObject() : GraphObject() {

  initialize();

}

// StreamObject built from SDE interface -- DL 9/10/96

StreamObject::StreamObject(STREAM st) : GraphObject() {

  initialize();
  this->read_from(st);

}

StreamObject::StreamObject(char *in_name_ptr, OP_ID in_id,
   OP_ID in_from, OP_ID in_to, int in_latency,
   int in_latency_unit,
   SplineObject *in_arc_ptr, BOOLEAN in_is_new,
   BOOLEAN in_is_state_variable) : GraphObject() {

  initialize();

  _id                  = in_id;
  _name_ptr            = dup_str(in_name_ptr);                    // kbm
  _from                = in_from;
  _from_ptr            = NULL;
  _to                  = in_to;
  _to_ptr              = NULL;
  _arc                 = *in_arc_ptr;  // deep copy
  latency(in_latency, in_latency_unit);
  _is_state_variable   = in_is_state_variable;

  _is_new              = in_is_new;

  set_text_dimensions();
  reset_handles_drawn_state();
}

StreamObject::~StreamObject() {

#ifdef GE_DEBUG
  //  printf("StreamObject Destructor called for: %s\n", _name_ptr);
#endif

  release();
}
```

```
StreamObject& StreamObject::operator=(StreamObject& src) {

  release();
  copy(&src);

  return *this;
}

STREAM StreamObject::clone(OP_LIST op_list) {

  STREAM st = (STREAM) malloc(sizeof(ST_NODE));

  write_to(st, op_list);

  return st;
}


//   Draws handles around the specified location.  Handles are
// drawn in exclusive-or mode to simplify erasing them without
// disturbing the underlying text.

void StreamObject::draw_handles(GC draw_context, int x1, int y1,
                                int x2, int y2) {

  XSetFunction(_display_ptr, draw_context, GXxor);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y2 - HANDLESIZE, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y1, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y2 - HANDLESIZE,
                 HANDLESIZE, HANDLESIZE);
  XSetFunction(_display_ptr, draw_context, GXcopy);
}

//   Draws text strings.

void StreamObject::draw_text(DRAW_STYLE style) {
  GC draw_context;

  int xN_pos, yN_pos,
      xL_pos, yL_pos;

  xN_pos = _x + _name_x;
  yN_pos = _y + _name_y;
  xL_pos = _x + _latency_x;
  yL_pos = _y + _latency_y;

  if(style == SOLID)
    draw_context = _graphics_context;
  else
    if(style == ERASE)
      draw_context = _erase_context;
  set_font(draw_context, _name_font);
  XDrawString(_display_ptr, _draw_window, draw_context,
              xN_pos, yN_pos, _name_ptr, strlen(_name_ptr));
  XDrawString(_display_ptr, *_drawing_area_pixmap, draw_context,
              xN_pos, yN_pos, _name_ptr, strlen(_name_ptr));
  if(_name_selected) {
    if((_name_handles_drawn == false) && (style == SOLID)) {
      draw_handles(_graphics_context, xN_pos  - HANDLESIZE,
                   yN_pos - _name_height - HANDLESIZE,
                   xN_pos + _name_width + HANDLESIZE,
                   yN_pos + HANDLESIZE);
      _name_handles_drawn = true;
    }
    else
      if((_name_handles_drawn == true) && (style == ERASE)) {
        draw_handles(_graphics_context, xN_pos  - HANDLESIZE,
                     yN_pos - _name_height - HANDLESIZE,
                     xN_pos + _name_width + HANDLESIZE,
                     yN_pos + HANDLESIZE);
        _name_handles_drawn = false;
      }
  }

  if(_latency != UNDEFINED_TIME) {                        // @1
    set_font(draw_context, _latency_font);
    XDrawString(_display_ptr, _draw_window, draw_context,
                xL_pos, yL_pos, _latency_string_ptr,
                strlen(_latency_string_ptr));
    XDrawString(_display_ptr, *_drawing_area_pixmap,
                draw_context, xL_pos, yL_pos,
                _latency_string_ptr,
                strlen(_latency_string_ptr));
    if(_latency_selected) {
      if((_latency_handles_drawn == false) && (style == SOLID)) {
        draw_handles(_graphics_context, xL_pos  - HANDLESIZE,
                     yL_pos - _latency_height - HANDLESIZE,
                     xL_pos + _latency_width + HANDLESIZE,
                     yL_pos + HANDLESIZE);
        _latency_handles_drawn = true;
```

```
      }
      else
        if((_latency_handles_drawn == true) &&
           (style == ERASE)) {
          draw_handles(_graphics_context,
                       xL_pos  - HANDLESIZE,
                       yL_pos - _latency_height - HANDLESIZE,
                       xL_pos + _latency_width + HANDLESIZE,
                       yL_pos + HANDLESIZE);
          _latency_handles_drawn = false;
        }
    }
  }
}

void StreamObject::erase_text() {

  draw_text(ERASE);
}

//   Writes the graphic attributes of the stream to disk.
// GE_STATUS StreamObject::write_to_disk(FILE *outfile)   //@2

//   Draws the stream.  The actual curved line is contained
// in a spline, which draws itself.

void StreamObject::draw(DRAW_STYLE style) {
  GC draw_context;
  EXTERN_STATUS status;

  if(_is_deleted == false) {
    if((_from_ptr != NULL) || (_to_ptr != NULL)) {
      if(style == SOLID)
        draw_context = _graphics_context;
      else
        if(style == ERASE)
          draw_context = _erase_context;
        else
          draw_context = _dotted_context;
      if(_from_ptr == NULL)
        status = FROM_EXTERNAL;
      else
        if(_to_ptr == NULL)
          status = TO_EXTERNAL;
        else
          status = NO_EXTERNAL;
      _arc.draw(this, draw_context, _graphics_context, style, status);
      draw_text(style);
      if((_from_ptr == NULL) || (_to_ptr == NULL)) {
        if(_from_ptr == NULL)
          _arc.set_extern_location(extern_location,
                                   FROM_EXTERNAL);
        else
          if(_to_ptr == NULL)
            _arc.set_extern_location(extern_location,
                                     TO_EXTERNAL);
        set_font(draw_context, _name_font);
        XDrawString(_display_ptr, _draw_window, draw_context,
                    extern_location.x, extern_location.y,
                    "External", strlen("External"));
        XDrawString(_display_ptr, *_drawing_area_pixmap,
                    draw_context, extern_location.x,
                    extern_location.y, "External",
                    strlen("External"));
      }
    }
  }
}

void StreamObject::erase() {

  draw(ERASE);
}

//   Returns true if the given coordinates are close to the
// line, or within the boundaries of the text strings.

BOOLEAN StreamObject::hit(int x, int y) {

  int xN_pos, yN_pos,
      xL_pos, yL_pos;

  xN_pos = _x + _name_x;
  yN_pos = _y + _name_y;
  xL_pos = _x + _latency_x;
  yL_pos = _y + _latency_y;

  if(_is_deleted)
    return false;
  else {
    if(_name_ptr != NULL) {
      if(strlen(_name_ptr) != 0)
        if(((x >= xN_pos) && (x <= (xN_pos + _name_width))) &&
           (y >= yN_pos - _name_height) && (y <= yN_pos)) {
          _name_selected = true;
          return true;
        }
    }
    if(_latency_string_ptr != NULL) {
      if(strlen(_latency_string_ptr) != 0)
        if(((x >= xL_pos) &&
            (x <= (xL_pos + _latency_width))) &&
           (y >= yL_pos - _latency_height) &&
           (y <= yL_pos)) {
          _latency_selected = true;
          return true;
```

```
        }
      }
      return _arc.hit(x, y);
    }
  }

  BOOLEAN StreamObject::over(int x, int y) {

    int xN_pos, yN_pos,
        xL_pos, yL_pos;

    xN_pos = _x + _name_x;
    yN_pos = _y + _name_y;
    xL_pos = _x + _latency_x;
    yL_pos = _y + _latency_y;

    if(_is_deleted)
      return false;
    else {
      if(_name_ptr != NULL) {
        if(strlen(_name_ptr) != 0)
          if(((x >= xN_pos) && (x <= (xN_pos + _name_width))) &&
             (y >= yN_pos - _name_height) && (y <= yN_pos)) {
            return true;
          }
      }
      if(_latency_string_ptr != NULL) {
        if(strlen(_latency_string_ptr) != 0)
          if(((x >= xL_pos) &&
              (x <= (xL_pos + _latency_width))) &&
             (y >= yL_pos - _latency_height) &&
             (y <= yL_pos)) {
            return true;
          }
      }
      return _arc.hit(x, y);
    }
  }

  //   Sets the _from_ptr and _to_ptr equal to the locations of
  // the from and to operators.  Simplifies getting properties
  // of the operators when needed.

  void StreamObject::set_object_ptrs(GraphObjectList *parent) {

    int x_pos, y_pos;

    _from_ptr = NULL;
    if (_from != UNDEFINED_OPNUM)
      _from_ptr = (OperatorObject *) parent->
                              target_object(OPERATOROBJECT, _from);
    _to_ptr   = NULL;
    if (_to != UNDEFINED_OPNUM)
      _to_ptr = (OperatorObject *) parent->
                              target_object(OPERATOROBJECT, _to);

    _arc.set_object_ptrs(_from_ptr, _to_ptr);
    _arc.set_offset_defaults(&x_pos, &y_pos);
    x(x_pos);
    y(y_pos);
    /* modified by M.T.Shing, 3/7/94 */
    /* set_default_text_location is only called if _name_x == NULL_VALUE */
    //  if (_name_x == NULL_VALUE)
    //   set_default_text_location();
  }

  //   Notifies the stream that the given object has been deleted.
  // If it's an operator connected to the stream, the stream
  // deletes itself.

  void StreamObject::delete_notify(CLASS_DEF class_type,
                                   OP_ID deleted_obj_id) {

    if((class_type == OPERATOROBJECT) &&
       ((_from == deleted_obj_id) || (_to == deleted_obj_id))) {
      erase();
    _is_deleted = true;
    }
  }

  void StreamObject::name(char *new_name) {

    free(_name_ptr);
    _name_ptr = dup_str(new_name);
    set_text_dimensions();
  }


  void StreamObject::replace_name(char *new_name) {

    free(_name_ptr);
    _name_ptr = dup_str(new_name);
    set_text_dimensions();
  }

  //   Notifies the stream that the given object has moved.  If
  // it's a connected object, the endpoints of the stream must
  // move.

  void StreamObject::move_notify(CLASS_DEF object_type, OP_ID object_id) {
    int x_pos, y_pos;

    if (object_type == OPERATOROBJECT) {
      if ((_from == object_id) || (_to == object_id)) {
        _arc.set_object_ptrs(_from_ptr, _to_ptr);
        _arc.set_offset_defaults(&x_pos, &y_pos);
```

```
      x(x_pos);
      y(y_pos);
    }
  }
  else
    if ((object_type == STREAMOBJECT) && (object_id == _id)) {
      _arc.set_object_ptrs(_from_ptr, _to_ptr);
      _arc.set_offset_defaults(&x_pos, &y_pos);
      x(x_pos);
      y(y_pos);
    }
}

void StreamObject::select() {

  erase();
  _is_selected = true;
  draw(SOLID);
}

void StreamObject::unselect() {

  erase();
  _is_selected = false;
  _name_selected = false;
  _latency_selected = false;
  draw(SOLID);
}

//   Returns true if any of the stream's handles enclose the
// given coordinates.

BOOLEAN StreamObject::hit_handle(int x, int y) {
  EXTERN_STATUS status;

  if(_from_ptr == NULL)
    status = FROM_EXTERNAL;
  else
    if(_to_ptr == NULL)
      status = TO_EXTERNAL;
    else
      status = NO_EXTERNAL;
  return _arc.hit_handle(x, y, status);
}

void StreamObject::set_default_text_location() {

  set_default_name_location();
  set_default_latency_location();

}

void StreamObject::set_default_name_location() {

  _name_x = 0;
  _name_y = -2;

}

void StreamObject::set_default_latency_location() {

  _latency_x = 0;
  _latency_y = 2 + font_text_height(_latency_font);

}

//   Sets latency from the value entered in the properties
// dialog box.

void StreamObject::latency(int latency, int unit) {
  int old_latency = _latency;

  _latency = latency;
  _latency_unit = unit;

  free(_latency_string_ptr);
  _latency_string_ptr = time_with_units(latency, unit);        // @4
  set_text_dimensions();
  if (old_latency == UNDEFINED_TIME)
    set_default_latency_location();
}

//   Sets values for the dimensions of the text strings.
//   _name_font and _latency_font must be set before calling!

void StreamObject::set_text_dimensions() {

  if(_name_ptr == NULL) {
    _name_width = 0;
    _name_height = 0;
  }
  else {
    _name_width = font_text_width(_name_font, _name_ptr);
    _name_height = font_text_height(_name_font);
  }
  if(_latency != UNDEFINED_TIME) {              // @1
    _latency_width = font_text_width(_latency_font,
                                     _latency_string_ptr);
    _latency_height = font_text_height(_latency_font);
  }
  else {
    _latency_width = 0;
    _latency_height = 0;
  }
}
```

```
void StreamObject::set_object_font(int font_id) {

  if(_name_selected)
    _name_font = font_id;
  else
    if(_latency_selected)
      _latency_font = font_id;
  set_text_dimensions();
}

//   Moves appropriate text strings the given amount.

void StreamObject::move_text(int x, int y) {

  erase_text();
  if (_name_selected) {
    _name_x += x;
    _name_y += y;
  }
  else
    if (_latency_selected) {
      _latency_x += x;
      _latency_y += y;
    }
  draw_text(SOLID);
}

//   Notifies the stream that the given object has been
// undeleted.  If it's an object connected to the stream,
// the stream may need to undelete itself.  The undeleted object
// could be the stream itself.

void StreamObject::undelete_notify(CLASS_DEF class_type,
                                   OP_ID deleted_obj_id) {

  if((class_type == STREAMOBJECT) &&
     (deleted_obj_id == _id)) {
    _is_deleted = false;
    _is_modified = true;
    _is_selected = false;
    _name_selected = false;
    _latency_selected = false;
  }
  else
    if((class_type == OPERATOROBJECT) &&
       (_is_deleted == true)) {
      if(_from == deleted_obj_id) {
        if(_to_ptr != NULL) {
          if(_to_ptr->is_deleted() == false) {
            _is_deleted = false;
            _is_modified = true;
            _is_selected = false;
            _name_selected = false;
            _latency_selected = false;
          }
        }
        else {
          _is_deleted = false;
          _is_modified = true;
          _is_selected = false;
          _name_selected = false;
          _latency_selected = false;
        }
      }
      else
        if(_to == deleted_obj_id) {
          if(_from_ptr != NULL) {
            if(_from_ptr->is_deleted() == false) {
              _is_deleted = false;
              _is_modified = true;
              _is_selected = false;
              _name_selected = false;
              _latency_selected = false;
            }
          }
          else {
            _is_deleted = false;
            _is_modified = true;
            _is_selected = false;
            _name_selected = false;
            _latency_selected = false;
          }
        }
    }
}

//   The handles_drawn states are used to prevent the handles
// from accidentally redrawing and erasing a handle.

void StreamObject::reset_handles_drawn_state() {

  _st_handles_drawn = false;
  _name_handles_drawn = false;
  _latency_handles_drawn = false;
  _arc.reset_handles_drawn();
}

int StreamObject::text_height() {

  if(_name_selected)
    return _name_height;
  else
    if(_latency_selected)
      return _latency_height;
    else
      return 0;
```

```
}

int StreamObject::text_width() {

  if(_name_selected)
    return _name_width;
  else
    if(_latency_selected)
      return _latency_width;
    else
      return 0;
}

//   Relocates the appropriate text strings at the given
// locations.

void StreamObject::text_locate(int x, int y) {

  if(_latency_selected) {
    _latency_x = (x - _x) - (_latency_width  / 2);
    _latency_y = (y - _y) + (_latency_height / 2);
  }
  else
    if(_name_selected) {
      _name_x = (x - _x) - (_name_width  / 2);
      _name_y = (y - _y) + (_name_height / 2);
    }
}

/*************************************************************************
 * build_st                                                        @5    *
 *                                                                       *
 * Builds a STREAM from a StreamObject.  NOTE:  This routine assumes     *
 * that the OperatorObjects have already been built (build_op) since     *
 * ge_interface.h includes OPERATOR pointers within the STREAM structure.*
 *                                                                       *
 *************************************************************************/
STREAM build_st(StreamObject *st, OP_LIST op_list) {

  STREAM geStPtr = (STREAM) malloc(sizeof(ST_LIST_NODE));

  geStPtr->id                = st->id();
  geStPtr->label             = dup_str(st->name());
  geStPtr->label_font        = st->name_font();
  geStPtr->label_x_offset    = st->name_x();
  geStPtr->label_y_offset    = st->name_y();
  geStPtr->from              = st->from();
  geStPtr->to                = st->to();
  if (st->spline_empty())
    geStPtr->arc             = NULL;
  else {
    if (geStPtr->from == EXTERNAL_VERTEX_NUM)
      geStPtr->arc             = st->write_spline_to_sde(FROM_EXTERNAL);
    else if (geStPtr->to == EXTERNAL_VERTEX_NUM)
      geStPtr->arc             = st->write_spline_to_sde(TO_EXTERNAL);
    else
      geStPtr->arc             = st->write_spline_to_sde(NO_EXTERNAL);
  }
  geStPtr->latency           = st->latency();
  geStPtr->latency_unit      = st->latency_unit();
  geStPtr->latency_font      = st->latency_font();
  geStPtr->latency_x_offset  = st->latency_x();
  geStPtr->latency_y_offset  = st->latency_y();
  geStPtr->stream_type_name  = dup_str(st->stream_type_name());
  geStPtr->state_initial_value = dup_str(st->state_initial_value());
  geStPtr->is_state_variable = st->is_state_variable();
  geStPtr->is_new            = st->is_new();
  geStPtr->is_modified       = st->is_modified();
  geStPtr->is_deleted        = st->is_deleted();

  return geStPtr;
}



/*************************************************************************
 * copy                                                            @5    *
 *                                                                       *
 * Builds a STREAM from a StreamObject.  NOTE:  This routine assumes     *
 * that the OperatorObjects have already been built (build_op) since     *
 * ge_interface.h includes OPERATOR pointers within the STREAM structure.*
 *                                                                       *
 *************************************************************************/
void StreamObject::copy(STREAM &st, OP_LIST op_list) {


  STREAM StPtr = (STREAM) malloc(sizeof(ST_NODE));
  st = StPtr;

  StPtr->id                  = _id;
  StPtr->label               = dup_str(_name_ptr);
  StPtr->label_font          = _name_font;
  StPtr->label_x_offset      = _name_x;
  StPtr->label_y_offset      = _name_y;
  StPtr->from                = _from;
  StPtr->to                  = _to;
  if (_arc.empty())
    StPtr->arc               = NULL;
  else {
    if (_from == EXTERNAL_VERTEX_NUM)
      StPtr->arc             = _arc.write_to_sde(FROM_EXTERNAL);
    else if (_to == EXTERNAL_VERTEX_NUM)
      StPtr->arc             = _arc.write_to_sde(TO_EXTERNAL);
    else
      StPtr->arc             = _arc.write_to_sde(NO_EXTERNAL);
  }
```

```
  StPtr->latency             = _latency;
  StPtr->latency_unit        = _latency_unit;
  StPtr->latency_font        = _latency_font;
  StPtr->latency_x_offset    = _latency_x;
  StPtr->latency_y_offset    = _latency_y;
  StPtr->stream_type_name    = dup_str(_stream_type_name);
  StPtr->state_initial_value = dup_str(_state_initial_value);
  StPtr->is_state_variable   = _is_state_variable;
  StPtr->is_new              = _is_new;
  StPtr->is_modified         = _is_modified;
  StPtr->is_deleted          = _is_deleted;
}

void StreamObject::write_to(STREAM StPtr, OP_LIST op_list ) {

  StPtr->id                  = _id;
  StPtr->label               = dup_str(_name_ptr);
  StPtr->label_font          = _name_font;
  StPtr->label_x_offset      = _name_x;
  StPtr->label_y_offset      = _name_y;
  StPtr->from                = _from;
  StPtr->to                  = _to;
  if (_arc.empty())
    StPtr->arc               = NULL;
  else {
    if (_from == EXTERNAL_VERTEX_NUM)
      StPtr->arc             = _arc.write_to_sde(FROM_EXTERNAL);
    else if (_to == EXTERNAL_VERTEX_NUM)
      StPtr->arc             = _arc.write_to_sde(TO_EXTERNAL);
    else
      StPtr->arc             = _arc.write_to_sde(NO_EXTERNAL);
  }
  StPtr->latency             = _latency;
  StPtr->latency_unit        = _latency_unit;
  StPtr->latency_font        = _latency_font;
  StPtr->latency_x_offset    = _latency_x;
  StPtr->latency_y_offset    = _latency_y;
  StPtr->stream_type_name    = dup_str(_stream_type_name);
  StPtr->state_initial_value = dup_str(_state_initial_value);
  StPtr->is_state_variable   = _is_state_variable;
  StPtr->is_new              = _is_new;
  StPtr->is_modified         = _is_modified;
  StPtr->is_deleted          = _is_deleted;
}


void StreamObject::read_from(STREAM st) {

  _id        = st->id;
  _name_ptr        = dup_str(st->label);                    // kbm
  _name_font        = st->label_font;
  _name_x        = st->label_x_offset;
  _name_y        = st->label_y_offset;
  _from        = st->from;
  _from_ptr           = NULL;
  _to        = st->to;
  _to_ptr             = NULL;
  _arc.build_from_sde(st->arc);
  _latency_font       = st->latency_font;
  latency(st->latency, st->latency_unit);    // must follow name & latency font
  _latency_x        = st->latency_x_offset;
  _latency_y        = st->latency_y_offset;
  _stream_type_name    = dup_str(st->stream_type_name);         // kbm
  _state_initial_value = dup_str(st->state_initial_value);      // kbm
  _is_state_variable   = st->is_state_variable;
  _is_new        = false;
  _is_modified         = false;
  _is_deleted        = false;
  _is_selected         = false;                                 // kbm
  _name_selected       = false;                                 // kbm
  _latency_selected    = false;                                 // kbm

  set_text_dimensions();                                        // kbm
  reset_handles_drawn_state();                                  // kbm

}

void StreamObject::release() {

#ifdef GE_DEBUG
  //  printf("StreamObject::release called for: %s\n", _name_ptr);
#endif

  free(_name_ptr);
  _name_ptr            = NULL;

  free(_latency_string_ptr);
  _latency_string_ptr  = NULL;

  free(_stream_type_name);
  _stream_type_name    = NULL;

  free(_state_initial_value);
  _state_initial_value = NULL;

  _from_ptr = NULL;
  _to_ptr   = NULL;

  _arc.clear();
}

void StreamObject::copy(StreamObject *src) {

  _id        = src->_id;
  _name_ptr        = dup_str(src->_name_ptr);
  _name_font        = src->_name_font;
```

```
  _name_x        = src->_name_x;
  _name_y        = src->_name_y;
  _from        = src->_from;
  _from_ptr            = src->_from_ptr;
  _to        = src->_to;
  _to_ptr              = src->_to_ptr;
  _arc        = src->_arc;
  _latency             = src->_latency;
  _latency_unit        = src->_latency_unit;
  _latency_font        = src->_latency_font;
  _latency_x        = src->_latency_x;
  _latency_y        = src->_latency_y;
  _stream_type_name    = dup_str(src->_stream_type_name);
  _state_initial_value = dup_str(src->_state_initial_value);
  _is_state_variable   = src->_is_state_variable;
  _is_new        = src->_is_new;
  _is_modified         = src->_is_modified;
  _is_deleted        = src->_is_deleted;

  _name_height         = src->_name_height;
  _name_width          = src->_name_width;
  _latency_string_ptr  = dup_str(src->_latency_string_ptr);
  _latency_height      = src->_latency_height;
  _latency_width       = src->_latency_width;

  _handle_selected     = src->_handle_selected;
  _is_selected         = src->_is_selected;
  _name_selected       = src->_name_selected;
  _latency_selected    = src->_latency_selected;
  _st_handles_drawn    = src->_st_handles_drawn;
  _name_handles_drawn  = src->_name_handles_drawn;
  _latency_handles_drawn = src->_latency_handles_drawn;

  name_location        = src->name_location;
  lat_location         = src->lat_location;
  extern_location      = src->extern_location;

}

void StreamObject::initialize() {

  _id                   = UNDEFINED_OPNUM;
  _name_ptr             = dup_str("");             // kbm
  _name_font            = _default_font;
  _name_x               = NULL_VALUE;
  _name_y               = NULL_VALUE;
  _from                 = UNDEFINED_OPNUM;
  _from_ptr             = NULL;
  _to                   = UNDEFINED_OPNUM;
  _to_ptr               = NULL;
  // _arc                 initialized by SplineObject constructor
  _latency              = UNDEFINED_TIME;          // @1
  _latency_unit         = MS;                      // @3
  _latency_string_ptr   = dup_str("");             // kbm
  _latency_font         = _default_font;
  _latency_x            = NULL_VALUE;
  _latency_y            = NULL_VALUE;
  _stream_type_name     = dup_str("");
  _state_initial_value  = dup_str("");
  _is_state_variable    = false;
  _is_new               = false;
  _is_modified          = false;
  _is_deleted           = false;

  _is_selected          = false;
  _name_selected        = false;
  _latency_selected     = false;
  set_text_dimensions();
  reset_handles_drawn_state();
}


void StreamObject::inherit_type(StreamObject *fromPtr) {

  char *fromString;

  fromString = fromPtr->stream_type_name();
  stream_type_name(fromString);
  free(fromString);

  fromString = fromPtr->state_initial_value();
  state_initial_value(fromString);
  free(fromString);

  is_state_variable(fromPtr->is_state_variable());

}
```

```
#include <Xm/MessageB.h>
#include "graph_object_list.h"
#include "operator_object.h"
#include "spline_object.h"
#include "stream_object.h"
#include <Xm/Xm.h>

#ifndef STREAM_PROPERTY_MENU_H
#define STREAM_PROPERTY_MENU_H
```

```
void stream_property_dialog(Widget parent,
    StreamObject *st_being_updated,
    int x, int y,
    GraphObjectList *graphic_list);

#endif /* STREAM_PROPERTY_MENU_H */
```

```
/*****************************************************************
 *
 * Project:      PSDL Editor
 * Assembly:     GUI (Graphics User Interface)
 * SubAssem:     Stream Properties
 *
 * Programmer:   Man-Tak Shing
 * Langauge:     C++
 *
 * Description:  This package is responsible for producing the pop-up
 *      window for displaying and maintaining the properties of a stream.
 *      stream_property_dialog is the main routine.  Several callback
 *      routines are also provided.
 *
 *      The StreamObject that is being updated is provided by
 *      st_being_updated, which is a global symbol.  It is expected to
 *      be set prior to calling stream_property_dialog.
 *
 *
 *
 * Modules:
 *
 * stream_name_cb
 *      initial_state_cb
 *      stream_type_cb
 *      state_query_cb
 *      latency_cb
 *      latency_unit_cb
 *      stream_ok_cb
 *      stream_cancel_cb
 *      stream_help_cb
 *      stream_property_dialog
 *
 *
 * Globals/Static:
 *
 *      st_being_updated
 *
 *      temp_st_info
 *
 *      initial_state
 *
 *      state_init_value
 *
 *      st_dialog
 *
 *      init_value_prefix
 *
 * History:
 *
 * Id   Date      Author        Change
 * @1   yy/mm/dd  Ken Moeller
 *
 *
 * Suggested Modifications:
 *
 *   Get rid of st_being_updated from graph_editor and pass it to
 *   stream_property_dialog as an argument.
 *
 *****************************************************************/

#include <stdlib.h>
#include <stream.h>

#include <X11/Xatom.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/Form.h>
#include <Xm/LabelG.h>
#include <Xm/List.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectioB.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/ToggleBG.h>

#include "action_area.h"
#include "ge_defs.h"
#include "ge_interface.h"
#include "ge_utilities.h"          // kbm
#include "gettopshell.h"
#include "graph_editor.h"
#include "graph_object_list.h"
#include "operator_object.h"
#include "spline_object.h"
#include "stream_object.h"
#include "warning.h"
#include "windows.h"

//extern StreamObject *st_being_updated /* = chosen stream  */;

//static STREAM temp_st_info = new(ST_NODE);
//static Widget initial_state;
//static Widget state_init_value = NULL;
//static Widget st_dialog;
//static char   *init_value_prefix = NULL;
//static BOOLEAN stream_latency_error;

StreamObject *temp_st_info;
Widget initial_state;
```

```
Widget st_dialog;
Widget state_init_value = NULL;
char   *init_value_prefix = NULL;
BOOLEAN stream_latency_error;
BOOLEAN stream_name_error;
BOOLEAN stream_type_error;
GraphObjectList *stDialogGraphicList;

// stream property menu

static void stream_name_cb(Widget w, XtPointer client_data,
                      XtPointer call_data) {

  Widget temp_w =  (Widget) client_data;
  char* text = NULL;

  text = XmTextFieldGetString(temp_w);

  if (!valid_id(text)) {
    warning(w, "Invalid stream name");
    update_status("Illegal stream name, correct or cancel:"
  " id ::= letter {alpha_numeric}",
  RING_BELL);
    XtFree(text);
    stream_name_error = true;
    return;
  }
  if (is_keyword(text, false)) {
    warning(w, "Stream name is a keyword");
    update_status("Stream name is a keyword, correct or cancel", RING_BELL);
    XtFree(text);
    stream_name_error = true;
    return;
  }
  if (text != NULL)
    temp_st_info->name(text);
  else
    temp_st_info->name("");

  stream_name_error = false;
  XtFree(text);
  return;
}

static void initial_state_ok_pushed(Widget w, XtPointer client_data,
    XtPointer call_data) {

  Widget text_w = (Widget)client_data;
  XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *)call_data;

  char *text = XmTextGetString(text_w);

  temp_st_info->state_initial_value(text);

  if ((text != NULL) && (*text != '\0')) {
    if (!state_init_value) {
      state_init_value =  XtVaCreateManagedWidget("state_init_value",
xmTextFieldWidgetClass, st_dialog,
XmNx, 150,
XmNy, 130,
                                XmNeditable, False,
                                XmNcursorPositionVisible, False,
NULL);
    }
    copy_str_eps(text, &init_value_prefix,19);
    XtVaSetValues(state_init_value, XmNvalue, init_value_prefix, NULL);
    free(init_value_prefix);  init_value_prefix = NULL;
  }
  else {
    if (state_init_value) {
      XtDestroyWidget(state_init_value);
      state_init_value = NULL;
    }
  }

  free(text);              text     = NULL;

  XtDestroyWidget(XtParent(XtParent(XtParent(w))));

  clear_status();
}

static void initial_state_cb(Widget w, XtPointer client_data,
     XtPointer call_data) {

  Widget    dialog, pane, rc, text_w, action_a;
  XmString  string;
  char      *description;

  static ActionAreaItem   action_items[] = {
    {"OK",     initial_state_ok_pushed, NULL                    },
    {"Cancel", close_dialog,          NULL                    },
    {"Help",   help_cb,               "initial_state.hlp"  }
  };

  dialog = XtVaCreatePopupShell ("dialog", xmDialogShellWidgetClass,
 XtParent(w),
 XmNtitle, "Stream Initial Value",
 XmNdeleteResponse, XmDESTROY,
 NULL);

  action_items[1].data = (XtPointer)dialog;   //Set cancel buttons client_data

  pane = XtVaCreateWidget("pane", xmPanedWindowWidgetClass, dialog,
  XmNsashWidth,   1,
  XmNsashHeight,  1,
```

```
  NULL);

  rc = XtVaCreateWidget("control_area", xmRowColumnWidgetClass, pane, NULL);
  string = XmStringCreateSimple("View or Edit Stream Initial Value");
  XtVaCreateManagedWidget("label", xmLabelGadgetClass, rc,
  XmNlabelString, string,
  NULL);
  XmStringFree(string);

  description = temp_st_info->state_initial_value();

  int     n = 0;
  Arg     args[10];
  XtSetArg(args[n], XmNrows,                  12); n++;
  XtSetArg(args[n], XmNcolumns,               70); n++;
  XtSetArg(args[n], XmNscrollVertical,        true); n++;
  XtSetArg(args[n], XmNscrollHorizontal,      true); n++;
  XtSetArg(args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
  XtSetArg(args[n], XmNeditable,              true); n++;
  XtSetArg(args[n], XmNcursorPositionVisible, true); n++;
  XtSetArg(args[n], XmNwordWrap,              true); n++;
  XtSetArg(args[n], XmNvalue,                 description); n++;
  text_w = XmCreateScrolledText(rc, "text-field", args, n);
  XtManageChild(text_w);
  //text_w = XtVaCreateManagedWidget("text-field", xmTextFieldWidgetClass,
  //      rc, NULL);

  XtAddCallback(text_w, XmNmodifyVerifyCallback, validate_text, NULL);

  XtManageChild(rc);

  //Set client data for the "OK" and "Cancel" buttons
  action_items[0].data = (XtPointer)text_w;

  action_a = CreateActionArea(pane, action_items, XtNumber(action_items));

  //XtAddCallback(text_w, XmNactivateCallback, activate_cb, action_a);

  XtManageChild(pane);
  free(description);
  XtPopup(dialog, XtGrabNone);


}

static void stream_type_cb(Widget w, XtPointer client_data,
   XtPointer call_data) {

  Widget temp_w =  (Widget) client_data;
  char* text = NULL;

  text = XmTextFieldGetString(temp_w);

  if ((*text != '\0') && !valid_type_name(text)) {
    warning(w, "Invalid stream type name (syntax or keyword)");
    update_status("Illegal stream type name, correct or cancel:"
  " type_name ::= id | id '[' type_decl ']'",
  RING_BELL);
    XtFree(text);
    stream_type_error = true;
    return;
  }

  if (text != NULL)
    temp_st_info->stream_type_name(text);
  else
    temp_st_info->stream_type_name("");

  stream_type_error = false;
  XtFree(text);
  return;
}


static void state_query_cb(Widget, XtPointer which,
                      XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  char* initial_str = NULL;

  if(state->set)
    temp_st_info->is_state_variable( (int) which);

  if (temp_st_info->is_state_variable()) {
    XtVaSetValues(initial_state, XmNsensitive, True, NULL);

    initial_str = temp_st_info->state_initial_value();

    if ((initial_str != NULL) && (*initial_str != '\0')) {
      if (!state_init_value) {
state_init_value =  XtVaCreateManagedWidget("state_init_value",
xmTextFieldWidgetClass, st_dialog,
XmNx, 150,
XmNy, 130,
XmNeditable, False,
                                        XmNcursorPositionVisible, False,
      NULL);
      }
      copy_str_eps(initial_str, &init_value_prefix,19);
      XtVaSetValues(state_init_value, XmNvalue, init_value_prefix, NULL);
      free(init_value_prefix);   init_value_prefix = NULL;
      }

  }
  else {
```

```
    XtVaSetValues(initial_state, XmNsensitive, False, NULL);
    if (state_init_value) {
      XtUnmanageChild(state_init_value);
      state_init_value = NULL;
    }
  }
  free(initial_str);
  return;
}


static void latency_cb(Widget w, XtPointer client_data,
                       XtPointer call_data) {

  char   buffer[INPUT_LINE_SIZE];
  Widget temp_w =  (Widget) client_data;
  char   *text = NULL;
  int    latency_value;

  text = XmTextFieldGetString(temp_w);

  if (!valid_integer_literal(text, &latency_value)) {
    latency_value = UNDEFINED_TIME;
    if (white_space(text))
      stream_latency_error = false;
    else {
      warning(w, "Illegal value for latency time");
      update_status("Illegal value for latency time (correct value or Cancel):"
    " time ::= digit {digit}",
    RING_BELL);
      stream_latency_error = true;
    }
  }
  else
    stream_latency_error = false;

  temp_st_info->latency(latency_value,
temp_st_info->latency_unit());

  // Display new value from latency
  if (latency_value != UNDEFINED_TIME) {
    sprintf(buffer, "%d", latency_value);
    XtVaSetValues(temp_w, XmNvalue, buffer, NULL);
  }

  XtFree(text);

  return;
}


static void latency_error_cb(Widget w, XtPointer client_data,
     XtPointer call_data) {

  Widget temp_w =  (Widget) client_data;

  if (stream_latency_error)
    XtVaSetValues(temp_w, XmNvalue, "", NULL);

  printf("latency_error_cb\n");
}


static void latency_unit_cb(Widget w, XtPointer which, XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if(state->set)
      temp_st_info->latency_unit( (int) which );

  return;
}


static void stream_ok_cb(Widget w, XtPointer client_data,
 XtPointer call_data) {

  StreamObject *st_being_updated;
  st_being_updated = (StreamObject *) client_data;

  if (stream_latency_error) return;
  if (stream_name_error)    return;
  if (stream_type_error)    return;

  st_being_updated->erase();

  *st_being_updated = *temp_st_info;

  XtDestroyWidget(XtParent(w));

  st_being_updated->draw(SOLID);

  stDialogGraphicList->propagate_stream(st_being_updated->id(), true);

  save_state(SAVE_REQUIRED);

  delete temp_st_info;

  clear_status();

  return;
}


static void stream_cancel_cb(Widget w, XtPointer client_data,
                       XtPointer call_data) {
```

```
  XtDestroyWidget(XtParent(w));

  delete temp_st_info;

  clear_status();

  return;
}


static void stream_help_cb(Widget w, XtPointer client_data,
                           XtPointer call_data) {
  help_cb(w, client_data, call_data);

  return;
}


void stream_property_dialog(Widget parent,
    StreamObject *st_being_updated,
    int x, int y,
    GraphObjectList *graphic_list) {

  Widget stream_name, radio_box, unit_box,
         stream_type, latency,
         ok_button, cancel_button, help_button;
  XmString button1, button2, t, init_value;
  Arg args[10];
  char* prompt, buffer[INPUT_LINE_SIZE];
  int initial_button, ac;

  stDialogGraphicList = graphic_list;
  temp_st_info = new StreamObject();
  state_init_value   = NULL;
  init_value_prefix = NULL;

  clear_status();
  stream_latency_error = false;
  stream_name_error    = false;
  stream_type_error    = false;

  if (st_being_updated == (StreamObject *)NULL) {
    prompt = dup_str("Bad Stream Pointer Passed To Stream Dialog");
    warning(parent, prompt);
    free(prompt);
  } else {  // Build dialog

    *temp_st_info = *st_being_updated;

    ac = 0;
    XtSetArg(args[ac], XmNheight,       300);   ac++;
    XtSetArg(args[ac], XmNwidth,        310);   ac++;
    XtSetArg(args[ac], XmNautoUnmanage, False); ac++;
    t = XmStringCreateSimple("Stream Property");
    XtSetArg(args[ac], XmNdialogTitle,  t);     ac++;

    st_dialog = XmCreateBulletinBoardDialog(GetTopShell(parent),
"Stream_Property", args, ac);

    XmStringFree(t);

    // create the stream label & text field

    char* name_str = NULL;

    prompt = dup_str("Stream Name:");
    XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, st_dialog,
    XmNx, 10,
    XmNy, 10,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);

    stream_name =  XtVaCreateManagedWidget("stream_name",
   xmTextFieldWidgetClass, st_dialog,
   XmNx, 150,
   XmNy, 10,
   NULL);
    name_str = temp_st_info->name();
    XtVaSetValues(stream_name, XmNvalue, name_str, NULL);
    free(name_str);
    free(prompt);

    // create the stream type label and text field

    char* type_str = NULL;

    prompt = dup_str("Stream Type:");
    XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, st_dialog,
    XmNx, 10,
    XmNy, 50,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    stream_type =  XtVaCreateManagedWidget("stream_type",
   xmTextFieldWidgetClass, st_dialog,
   XmNx, 150,
   XmNy, 50,
   NULL);

    type_str = temp_st_info->stream_type_name();
    XtVaSetValues(stream_type, XmNvalue, type_str, NULL);
    free(type_str);
    free(prompt);

    // create a state_stream query label
    prompt = dup_str("Is a state stream?");
```

```
    XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, st_dialog,
    XmNx, 10,
    XmNy, 90,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    free(prompt);


    // create radio-box for state stream
    button1 = XmStringCreateSimple("No");
    button2 = XmStringCreateSimple("Yes");
    if (temp_st_info->is_state_variable())
      initial_button = 1;
    else
      initial_button = 0;
    radio_box = XmVaCreateSimpleRadioBox(st_dialog, "radio_box",
initial_button, state_query_cb,
XmVaRADIOBUTTON, button1, NULL, NULL, NULL,
XmVaRADIOBUTTON, button2, NULL, NULL, NULL,
XmNx, 150,
XmNy, 90,
XmNorientation, XmHORIZONTAL,
NULL);
    XmStringFree(button1);
    XmStringFree(button2);
    XtManageChild(radio_box);

    // create the initial state value label and text field

    char* initial_str = NULL;

    initial_state
      = XtVaCreateManagedWidget(dup_str("State Initial Value"),
xmPushButtonGadgetClass, st_dialog,
XmNx, 10,
XmNy, 130,
NULL);

    XtAddCallback(initial_state, XmNactivateCallback,
  initial_state_cb, (XtPointer)NULL);

    if (temp_st_info->is_state_variable())
      XtVaSetValues(initial_state, XmNsensitive, True, NULL);
    else
      XtVaSetValues(initial_state, XmNsensitive, False, NULL);

    initial_str = temp_st_info->state_initial_value();

    if ((temp_st_info->is_state_variable()) &&
(initial_str != NULL) && (*initial_str != '\0')) {
      state_init_value =  XtVaCreateManagedWidget("state_init_value",
xmTextFieldWidgetClass, st_dialog,
XmNx, 150,
XmNy, 130,
                                XmNeditable, False,
                                XmNcursorPositionVisible, False,
NULL);
      copy_str_eps(initial_str, &init_value_prefix,19);
      XtVaSetValues(state_init_value, XmNvalue, init_value_prefix, NULL);
      free(init_value_prefix);  init_value_prefix = NULL;
    }
    else
      state_init_value = NULL;

    free(initial_str);

    // create the latency label and text field
    prompt = dup_str("Latency:");
    XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, st_dialog,
    XmNx, 10,
    XmNy, 170,
    XmNalignment, XmALIGNMENT_BEGINNING,
    NULL);
    latency =  XtVaCreateManagedWidget("latency",
    xmTextFieldWidgetClass, st_dialog,
    XmNx, 150,
    XmNy, 170,
    XmNcolumns, 10,
    NULL);
    if (temp_st_info->latency() != UNDEFINED_TIME) {
      sprintf(buffer, "%d", temp_st_info->latency());
      XtVaSetValues(latency, XmNvalue, buffer, NULL);
    }
    free(prompt);

    unit_box = time_unit_menu(st_dialog, "unit_box",
    temp_st_info->latency_unit(), latency_unit_cb,
    220,170);
    XtManageChild(unit_box);

    XtVaCreateManagedWidget("separator", xmSeparatorWidgetClass, st_dialog,
    XmNy,     220,
    XmNwidth, 310,
    NULL);

    // create the OK, Cancel, Help buttons
    ac = 0;
    XtSetArg(args[ac], XmNx, 20); ac++;
    XtSetArg(args[ac], XmNy, 240); ac++;
    XtSetArg(args[ac], XmNlabelString,
     XmStringCreateSimple("OK"));
    ok_button = XmCreatePushButton(st_dialog, "   OK   ", args, ac);
    XtManageChild(ok_button);

    ac = 0;
    XtSetArg(args[ac], XmNx, 125); ac++;
```

```
    XtSetArg(args[ac], XmNy, 240); ac++;
    XtSetArg(args[ac], XmNlabelString,
     XmStringCreateSimple("Cancel"));
    cancel_button = XmCreatePushButton(st_dialog, " Cancel ", args, ac);
    XtManageChild(cancel_button);

    ac = 0;
    XtSetArg(args[ac], XmNx, 230); ac++;
    XtSetArg(args[ac], XmNy, 240); ac++;
    XtSetArg(args[ac], XmNlabelString,
     XmStringCreateSimple("Cancel"));
    help_button = XmCreatePushButton(st_dialog, "  HELP  ", args, ac);
    XtManageChild(help_button);

//    XtAddCallback(latency, XmNlosingFocusCallback, latency_cb,
//   (XtPointer)latency);
    XtAddCallback(ok_button, XmNactivateCallback,
  stream_name_cb, (XtPointer)stream_name);
    XtAddCallback(ok_button, XmNactivateCallback, stream_type_cb,
  (XtPointer)stream_type);
    XtAddCallback(ok_button, XmNactivateCallback, latency_cb,
  (XtPointer)latency);
```

```
    XtAddCallback(ok_button, XmNactivateCallback, stream_ok_cb,
  (XtPointer) st_being_updated);

    XtAddCallback(cancel_button, XmNactivateCallback, stream_cancel_cb,
  (XtPointer) st_dialog);

    XtAddCallback(help_button, XmNactivateCallback, stream_help_cb,
  (XtPointer) "stream_property.hlp");

    XtManageChild(st_dialog);

    XtPopup(GetTopShell(st_dialog), XtGrabNone);

    XmProcessTraversal(parent, XmTRAVERSE_CURRENT);
  }

  return;
}
```

```
#include<Xm/List.h>

#ifndef TIMER_TOOL_H
#define TIMER_TOOL_H 1

void timer_tool_add_cb (Widget w, XtPointer, XtPointer);
void timer_tool_del_cb (Widget w, XtPointer, XtPointer);
void timer_tool_edit_cb (Widget w, XtPointer, XtPointer);

#endif
```

```
/* ************************************************************

Name:     timer_tool.C
Author:   Lange and Anunciado
Program:  graph_editor
Remarks:

History:
  @1   96/09/29  Ken Moeller
       Migration from Motif 1.2 to Motif 1.1.

************************************************************ */

#include "ge_defs.h"
#include "ge_interface.h"
#include "timer_tool.h"
#include "ge_utilities.h"
#include "graph_editor.h"
#include "action_area.h"
#include "warning.h"
#include <Xm/SelectioB.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>


void read_timer_id(Widget widget, XtPointer client_data,
   XtPointer call_data) {

  Widget list_w = (Widget)client_data;
  XmSelectionBoxCallbackStruct *cbs =
    (XmSelectionBoxCallbackStruct *)call_data;
  char     *text, *newtext;
  int     u_bound;
  XmString *strlist;

  //check to make sure entry is not null
  // if (XmStringGetLtoR(cbs->value, XmFONTLIST_DEFAULT_TAG, &newtext)) // @1
  if (XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET, &newtext)) {// @1
    if (newtext && *newtext){

      if (!valid_id(newtext)) {
warning(widget, "Invalid timer ID");
update_status("Illegal ID: id ::= letter {alpha_numeric}",
      RING_BELL);
return;
      }
      if (is_keyword(newtext, false)) {
warning(widget, "Timer ID is a keyword");
update_status("Timer ID is a keyword, change or cancel",
                    RING_BELL);
return;
      }
      XmListAddItemUnselected(list_w, cbs->value, 0);
    }
  }
  XtDestroyWidget(widget);
}

void edit_timer_id(Widget widget, XtPointer client_data,
                   XtPointer call_data) {

  Widget list_w = (Widget)client_data;
  XmSelectionBoxCallbackStruct *cbs = (XmSelectionBoxCallbackStruct *)
                                      call_data;
  char     *text, *newtext;
  int     u_bound;
  XmString *strlist, new_items /* [1] */;  // @1
  int     i, *pos_list, pos_cnt;

  //check to make sure entry is not null
  // if (XmStringGetLtoR(cbs->value, XmFONTLIST_DEFAULT_TAG, &newtext)) // @1
  if (XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET, &newtext)) {// @1
    if (newtext && *newtext){

      if (!valid_id(newtext)) {
warning(widget, "Invalid timer ID");
update_status("Illegal ID: id ::= letter {alpha_numeric}",
      RING_BELL);
return;
      }
      if (is_keyword(newtext, false)) {
warning(widget, "Timer ID is a keyword");
update_status("Timer ID is a keyword, change or cancel",
                    RING_BELL);
return;
      }

      XmListAddItemUnselected(list_w, cbs->value, 0);
    }
  }
  XtDestroyWidget(widget);
}

void timer_tool_add_cb(Widget w, XtPointer client_data,
                       XtPointer call_data) {

  Widget   dialog;
  Widget   list_w = (Widget)client_data;
  XmString t = XmStringCreateSimple("Enter New Timer ID");     // @1
  Arg      args[5];
  int      n = 0;

  XtSetArg(args[n], XmNselectionLabelString, t); n++;
  XtSetArg(args[n], XmNautoUnmanage, false);     n++;
  dialog = XmCreatePromptDialog(w, "prompt", args, n);
  XmStringFree(t);
```

```
  XtAddCallback(dialog, XmNokCallback, read_timer_id, list_w);
  XtAddCallback(dialog, XmNcancelCallback, dlg_callback, NULL);
  XtSetSensitive(XmSelectionBoxGetChild(dialog, XmDIALOG_HELP_BUTTON),
                 false);
  XtManageChild(dialog);

  XtPopup(XtParent(dialog), XtGrabNone);

}

void timer_tool_del_cb(Widget widget, XtPointer client_data,
                       XtPointer call_data) {

  Widget dialog;
  Widget list_w = (Widget)client_data;
  int    *pos_list, pos_cnt;

  if (!XmListGetSelectedPos(list_w, &pos_list, &pos_cnt)) {
    warning(widget, "Nothing Selected");
    return;
  }

  XmListDeletePos(list_w, pos_list[0]);
  return;

}

void timer_tool_edit_cb(Widget w, XtPointer client_data,
                        XtPointer call_data) {

  Widget   dialog;
  Widget   list_w = (Widget)client_data;
  XmString t = XmStringCreateSimple("Enter Revised Timer ID");     // @1
  Arg      args[5];
  int      n = 0, *pos_list, pos_cnt;

  if (!XmListGetSelectedPos(list_w, &pos_list, &pos_cnt)) {
    warning(w, "Nothing Selected");
    return;
  }
  XtSetArg(args[n], XmNselectionLabelString, t); n++;
  XtSetArg(args[n], XmNautoUnmanage, false);     n++;
  dialog = XmCreatePromptDialog(w, "prompt", args, n);
  XmStringFree(t);

  XtAddCallback(dialog, XmNokCallback, edit_timer_id, list_w);
  XtAddCallback(dialog, XmNcancelCallback, dlg_callback, NULL);
  XtSetSensitive(XmSelectionBoxGetChild(dialog, XmDIALOG_HELP_BUTTON),
                 false);
  XtManageChild(dialog);

}

void timer_tool_help_cb(Widget w, XtPointer, XtPointer) {
}
```

```
#include <iostream.h>
#include <Xm/RowColumn.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>

#ifndef WARNING_H
#define WARNING_H 1

void dlg_callback(Widget, XtPointer, XtPointer);
void warning(Widget, char *s);

#endif
```

```
/* ************************************************************

Name:     warning.C
Author:   Lange and Anunciado
Program:  graph_editor
Remarks:

History:
  @1   96/09/29  Ken Moeller
       Migration from Motif 1.2 to Motif 1.1.

************************************************************ */

#include "warning.h"
void dlg_callback(Widget dialog, XtPointer client_data, XtPointer call_data) {

XtPopdown(XtParent(dialog));

}


void warning(Widget widget, char *string) {

Widget dialog;
XmString t;
Arg args[5];
int n = 0;

// XmString ok = XmStringCreateLocalized("OK");  // @1
XmString ok = XmStringCreateSimple("OK");        // @1
XtSetArg(args[n], XmNautoUnmanage,   False); n++;
XtSetArg(args[n], XmNcancelLabelString, ok); n++;

dialog = XmCreateInformationDialog(widget, "notice", args, n);
XtAddCallback(dialog, XmNcancelCallback, dlg_callback, NULL);

XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_OK_BUTTON));
XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON));

// t = XmStringCreateLocalized(string);  // @1
t = XmStringCreateSimple(string);        // @1
XtVaSetValues(dialog,
XmNmessageString, t,
XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
NULL);
XmStringFree(t);
XtManageChild(dialog);
XtPopup(XtParent(dialog), XtGrabExclusive);
}
```

```
/* Written by Dan Heller and Paula Ferguson.
 * Copyright 1994, O'Reilly & Associates, Inc.
 * Permission to use, copy, and modify this program without
 * restriction is hereby granted, as long as this copyright
 * notice appears in each copy of the program source code.
 * This program is freely distributable without licensing fees and
 * is provided without guarantee or warrantee expressed or implied.
 * This program is -not- in the public domain.
 */

/* ask_user.c -- the user is presented with two pushbuttons.
 * The first creates a file (/tmp/foo) and the second removes it.
 * In each case, a dialog pops up asking for verification of the action.
 *
 * This program is intended to demonstrate an advanced implementation
 * of the AskUser() function.  This time, the function is passed the
 * strings to use for the OK button and the Cancel button as well as
 * the button to use as the default value.
 */

#ifndef windows_h
#define windows_h 1

#include <malloc.h>
#include <Xm/DialogS.h>
#include <Xm/SelectioB.h>
#include <Xm/RowColumn.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include "ge_utilities.h"

#define YES    1
#define NO     2
#define CANCEL 3
#define OK     4

#define BTN1   1
#define BTN2   2
#define BTN3   3

// Save_indicator status
#define NOT_MODIFIED  0
#define SAVE_REQUIRED 1

/* Generalize the question/answer process by creating a data structure
 * that has the necessary labels, questions and everything needed to
 * execute a command.
 */
typedef struct {
  char *label;     // label for pushbutton used to invoke cmd
  char *question;  // question for dialog box to confirm cmd
  char *btn1;      // label for first button
  char *btn2;      // label for second button
  char *btn3;      // label for third button
  int   dflt;      // which should be the default answer
} Quest_Script;

enum print_opt { Snd_to_Prt, Snd_to_File };

typedef struct {
  print_opt  op;
  char      *printer;     // Name of printer
  char      *file;        // Name of screen dump file
  int        answer;      // Command that was requested (OK, Cancel)
} PrintBuf;


int AskUser(XtAppContext app, Widget parent, Quest_Script scr);
void response(Widget widget, XtPointer client_data, XtPointer call_data);

void AskPrint(XtAppContext app, Widget parent, PrintBuf *prn);
void PrintResponse(Widget widget, XtPointer client_data, XtPointer call_data);
Widget PostNotice(Widget widget, char *str);

Widget time_unit_menu(Widget parent, char *title,
                      int units, XtCallbackProc cb_routine,
      int x, int y);

void  copy_str_eps(char *src, char **dest_ptr, int str_size);
void  id_list_str_eps(ID_LIST src, char **dest_ptr, int str_size);

void validate_text(Widget widget, XtPointer client_data, XtPointer call_data);

#endif
```

```
#include "windows.h"
#include "warning.h"
#include "graph_editor.h"

/* Written by Dan Heller and Paula Ferguson.
 * Copyright 1994, O'Reilly & Associates, Inc.
 * Permission to use, copy, and modify this program without
 * restriction is hereby granted, as long as this copyright
 * notice appears in each copy of the program source code.
 * This program is freely distributable without licensing fees and
 * is provided without guarantee or warrantee expressed or implied.
 * This program is -not- in the public domain.
 */

static Widget prt_dialog;               // static to avoid multiple creation
static PrintBuf temp_PrintCmd;
extern Widget drawing_a;

/*****************************************************************************
 * AskUser() -- a generalized routine that asks the user a question
 * and returns a response.
 *****************************************************************************/
int AskUser(XtAppContext app, Widget parent, Quest_Script scr)
{
    static Widget dialog;               // static to avoid multiple creation
    XmString text, Lbl1BTN1, Lbl1BTN2, Lbl1BTN3;
    static int answer;
    int default_ans;

    if (!dialog) {
        dialog = XmCreateQuestionDialog(parent, "dialog", NULL, 0);
        XtVaSetValues(dialog,
            XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
            NULL);

        XtAddCallback(dialog, XmNokCallback,     response, &answer);
        XtAddCallback(dialog, XmNcancelCallback, response, &answer);
        XtAddCallback(dialog, XmNhelpCallback,   response, &answer);
    }

    answer = 0;
    text    = XmStringCreateSimple(scr.question);
    Lbl1BTN1 = XmStringCreateSimple(scr.btn1);
    Lbl1BTN2 = XmStringCreateSimple(scr.btn2);
    Lbl1BTN3 = XmStringCreateSimple(scr.btn3);
    if (scr.dflt == BTN1)
      default_ans = XmDIALOG_OK_BUTTON;
    else if (scr.dflt == BTN2)
      default_ans = XmDIALOG_CANCEL_BUTTON;
    else
      default_ans = XmDIALOG_HELP_BUTTON;

    XtVaSetValues(dialog,
        XmNmessageString,      text,
        XmNokLabelString,      Lbl1BTN1,
        XmNcancelLabelString,  Lbl1BTN2,
XmNhelpLabelString,    Lbl1BTN3,
        XmNdefaultButtonType,  default_ans,
        NULL);

    XmStringFree(text);
    XmStringFree(Lbl1BTN1);
    XmStringFree(Lbl1BTN2);
    XmStringFree(Lbl1BTN3);

    XtManageChild(dialog);
    XtPopup(XtParent(dialog), XtGrabNone);

    while (answer == 0)
      XtAppProcessEvent(app, XtIMAll);

    XtPopdown(XtParent(dialog));
    /* make sure the dialog goes away before returning. Sync with server
     * and update the display.
     */
    XSync(XtDisplay(dialog), 0);
    XmUpdateDisplay(parent);

    return answer;
}


/*****************************************************************************
 * response() --The user made some sort of response to the
 * question posed in AskUser().  Set the answer (client_data)
 * accordingly.
 *****************************************************************************/
void response(Widget widget, XtPointer client_data, XtPointer call_data)
{
    int *answer = (int *) client_data;
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *) call_data;

    if (cbs->reason == XmCR_OK)
      *answer = BTN1;
    else if (cbs->reason == XmCR_CANCEL)
      *answer = BTN2;
    else
      *answer = BTN3;
}


/*****************************************************************************
 * print_opt_selection_cb() -- displays the proper prompt for the printer
 * device (Printer or File name requested).
 *****************************************************************************/
```

```
void print_opt_selection_cb(Widget opt, XtPointer which,
    XtPointer call_data) {

    XmString label;
    XmString prn_str;

    if ((print_opt) which == Snd_to_Prt) {
      label   = XmStringCreateSimple("Printer Name:");
      prn_str = XmStringCreateSimple(temp_PrintCmd.printer);
    }
    else {
      label = XmStringCreateSimple("File Name:");
      prn_str = XmStringCreateSimple(temp_PrintCmd.file);
    }

    temp_PrintCmd.op = (print_opt) which;

    XtVaSetValues(prt_dialog,
        XmNselectionLabelString, label,
XmNtextString,           prn_str,
        NULL);

    XmStringFree(label);

}


/*****************************************************************************
 * AskPrint() -- Provide the user with a printer popup with the option
 * to enter a printer name. Returns both answer and printer in PrintCmd.
 *****************************************************************************/
void AskPrint(XtAppContext app, Widget parent, PrintBuf *prn)
{
    static Widget opt;
    temp_PrintCmd.printer = dup_str(prn->printer);
    temp_PrintCmd.file    = dup_str("");
    temp_PrintCmd.op      = prn->op;

    XmString opt_lbl  = XmStringCreateSimple("Print to:");
    XmString opt_prt  = XmStringCreateSimple("Printer");
    XmString opt_file = XmStringCreateSimple("File");
    XmString prn_str;
    XmString label;

    if (temp_PrintCmd.op == Snd_to_Prt) {
      label   = XmStringCreateSimple("Printer Name:");
      prn_str = XmStringCreateSimple(temp_PrintCmd.printer);
    }
    else {
      label = XmStringCreateSimple("File Name:");
      prn_str = XmStringCreateSimple(temp_PrintCmd.file);
    }

    if (!prt_dialog) {
        prt_dialog = XmCreatePromptDialog(parent, "Printer", NULL, 0);
        XtVaSetValues(prt_dialog,
            XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
            NULL);

opt = XmVaCreateSimpleRadioBox(prt_dialog, "radio_box",
       temp_PrintCmd.op,
       print_opt_selection_cb,
       XmVaRADIOBUTTON, opt_prt,  0, NULL, NULL,
       XmVaRADIOBUTTON, opt_file, 0, NULL, NULL,
       XmNorientation, XmHORIZONTAL,
       NULL);

        XtAddCallback(prt_dialog, XmNokCallback,
      PrintResponse, &temp_PrintCmd);
        XtAddCallback(prt_dialog, XmNcancelCallback,
      PrintResponse, &temp_PrintCmd);
        XtAddCallback(prt_dialog, XmNhelpCallback,
      help_cb,       "print.hlp");
    }

    XtVaSetValues(prt_dialog,
        XmNselectionLabelString, label,
        XmNdefaultButtonType,    XmDIALOG_OK_BUTTON,
XmNtextString,           prn_str,
        NULL);


    XmStringFree(opt_lbl);
    XmStringFree(opt_prt);
    XmStringFree(opt_file);
    XmStringFree(label);
    XmStringFree(prn_str);

    XtManageChild(opt);
    XtManageChild(prt_dialog);
    XtPopup(XtParent(prt_dialog), XtGrabNone);

    temp_PrintCmd.answer = 0;
    while (temp_PrintCmd.answer == 0)
      XtAppProcessEvent(app, XtIMAll);

    XtPopdown(XtParent(prt_dialog));
    /* make sure the dialog goes away before returning. Sync with server
     * and update the display.
     */
    XSync(XtDisplay(prt_dialog), 0);
    XmUpdateDisplay(parent);

    prn->answer = temp_PrintCmd.answer;
    prn->op     = temp_PrintCmd.op;      // do not cancel op picked
    if (prn->answer == OK) {
```

```
      free(prn->file);     prn->file    = dup_str(temp_PrintCmd.file);
      free(prn->printer);  prn->printer = dup_str(temp_PrintCmd.printer);
    }

    free(temp_PrintCmd.file);       temp_PrintCmd.file    = NULL;
    free(temp_PrintCmd.printer);    temp_PrintCmd.printer = NULL;
}

/*****************************************************************************
 * PrintResponse() --The user made some sort of response to the
 * question posed in AskUser().  Set the answer (client_data)
 * accordingly.
 *****************************************************************************/
void PrintResponse(Widget widget, XtPointer client_data, XtPointer call_data)
{
    PrintBuf *PrintCmdPtr = (PrintBuf *) client_data;
    char     *name;

    XmSelectionBoxCallbackStruct *cbs =
      (XmSelectionBoxCallbackStruct *) call_data;

    if (cbs->reason == XmCR_OK) {
      XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET, &name);
      if (PrintCmdPtr->op == Snd_to_Prt) {
free(PrintCmdPtr->printer);
PrintCmdPtr->printer = dup_str(name);
      }
      else {
free(PrintCmdPtr->file);
PrintCmdPtr->file = dup_str(name);
      }
      PrintCmdPtr->answer = OK;
    }
    else if (cbs->reason == XmCR_CANCEL)
      PrintCmdPtr->answer = CANCEL;
    else
      PrintCmdPtr->answer = CANCEL;
}

/*****************************************************************************
 * PostNotice()
 *
 *****************************************************************************/
Widget PostNotice(Widget widget, char *str) {

  Widget   dialog;
  XmString Xstr;
  Arg      args[5];
  int      n = 0;

  if (str) {       // only if a string was provided
    Xstr   = XmStringCreateSimple(str);
    XtSetArg(args[n], XmNautoUnmanage, False);  n++;

    dialog = XmCreateInformationDialog(widget, "notice", args, n);
    XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_OK_BUTTON));
    XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_CANCEL_BUTTON));
    XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON));

    XtVaSetValues(dialog,
  XmNmessageString, Xstr,
  XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
  NULL);
    XmStringFree(Xstr);
    XtManageChild(dialog);
    XtPopup(XtParent(dialog), XtGrabExclusive);
    return dialog;
  }
  return NULL;
}

/*****************************************************************************
 * time_unit_menu() -- Creates a SimpleOptionMenu with the time units
 * understood by PSDL.
 *****************************************************************************/
Widget time_unit_menu(Widget parent, char *title,
                      int units, XtCallbackProc cb_routine,
      int x, int y) {

  XmString microsec, ms, sec, min, hour;

  Widget menu_widget;

  microsec = XmStringCreateSimple("microsec");
  ms       = XmStringCreateSimple("ms");
  sec      = XmStringCreateSimple("sec");
  min      = XmStringCreateSimple("min");
  hour     = XmStringCreateSimple("hour");
            .
  menu_widget = XmVaCreateSimpleOptionMenu(parent, title,
          NULL, NULL, units, cb_routine,
          XmVaPUSHBUTTON, microsec, NULL, NULL, NULL,
          XmVaPUSHBUTTON, ms,       NULL, NULL, NULL,
          XmVaPUSHBUTTON, sec,      NULL, NULL, NULL,
          XmVaPUSHBUTTON, min,      NULL, NULL, NULL,
          XmVaPUSHBUTTON, hour,     NULL, NULL, NULL,
          XmNx, x,
          XmNy, y,
          NULL);

    XmStringFree(microsec);
    XmStringFree(ms);
    XmStringFree(sec);
    XmStringFree(min);
    XmStringFree(hour);
```

```
    return menu_widget;
}

/*****************************************************************************
 * copy_str_eps() -- Produces a copy of src which is at most str_size long.
 * If longer than str_size, the string will be truncated and "..." added
 * to the end of the string.  str_size should be at least 3.
 * NOTE:  No memory allocated to *dest_ptr will be recovered.  Caller should
 * verify that memory is recovered before calling.
 *****************************************************************************/
void  copy_str_eps(char *src, char **dest_ptr, int str_size) {

  int pos;
  char *new_line;
  char *dest;

  if (str_size < 3) str_size = 3;

  dest = (char *) malloc(str_size+1);
  *dest_ptr = dest;

  if (strlen(src)>(str_size)) {
    strncpy(dest,src,str_size-3);
    *(dest+str_size-3) = '.';
    *(dest+str_size-2) = '.';
    *(dest+str_size-1) = '.';
    *(dest+str_size) = '\0';
  }
  else
    strcpy(dest,src);

  new_line = strchr(dest,'\n');
  if (new_line)
    *new_line = '\0';
}



/*****************************************************************************
 * id_list_str_eps() -- Produces a copy of src which consists of each id
 * from the ID_LIST, separated by commas up to the length of str_size.
 * If the ID_LIST is longer than str_size, the string will be truncated and
 * "..." added to the end of the string.  str_size should be at least 3.
 * NOTE:  No memory allocated to *dest_ptr will be recovered.  Caller should
 * verify that memory is recovered before calling.
 *****************************************************************************/
void  id_list_str_eps(ID_LIST src, char **dest_ptr, int str_size) {

  char    *dest;
  char    *add_point;
  ID_LIST  idPtr;
  int      idStrLen, copyLen,curStrLen;

  char     comma[] = ", ";
  char     eps[]   = "...";


  if (src == NULL)
    dest = dup_str("");
  else {

    if (str_size < 3) str_size = 3;
    dest = (char *) malloc(str_size+1);
    *dest = '\0';

    idPtr     = src;
    curStrLen = 0;
    while ((idPtr != NULL) && (curStrLen < str_size)) {

      idStrLen  = strlen(idPtr->id);
      if ((idStrLen <= (str_size - curStrLen - 5)) ||
  ((idStrLen <= (str_size - curStrLen)) && (idPtr->next == NULL))) {
strcat(dest, idPtr->id);

if (idPtr->next != NULL) {
  strcat(dest, (char*) &comma);
}

idPtr = idPtr->next;   /* set up for next loop */
      }
      else {
copyLen = str_size - curStrLen - 3;
if (idStrLen < copyLen) copyLen = idStrLen;
strncat(dest, idPtr->id, copyLen);
strcat(dest, (char*) &eps);

idPtr = NULL;  /* break out of loop */
      }
      curStrLen = strlen(dest);
    }
  }

  *dest_ptr = dest;

}


/*****************************************************************************
 * validate_text() -- "}" is the only character not allowed in a PSDL
 * text field.
 *****************************************************************************/
void validate_text(Widget widget, XtPointer client_data,
   XtPointer call_data) {

  XmTextVerifyCallbackStruct *cbs =
    (XmTextVerifyCallbackStruct *) call_data;
```

```
if (cbs->text->ptr == NULL)
  return;

if (strchr(cbs->text->ptr,'}')) {
  warning(drawing_a,"'}' is not allowed in this field.");
  update_status("Text Field: Any printable character except for '}'",
    RING_BELL);
    cbs->doit = False;
  }
}
```

Error Messages Popup Window

This window presents all syntax and sematic warnings and errors detected by the PSDL Editor. The operator in which the error was detected is indicated under the "Current" column. The "Parent" column indicates the respective parent operator within the PSDL hierarchical operator tree.

From this window, direct access is provided to the operator in which the error was detected or to that operator's parent operator. Prior to accessing the desired operator, the associated error message must be selected. This is accomplished by depressing the left-mouse button with the cursor over the desired error message.

User options:

Close	-- Removes the Error Messages popup window.
Goto Parent	-- Calls up the parent operator associated with the selected error message.
Goto Current	-- Calls up the operator associated with the selected error message.
Help	-- This display.

Exceptions:

Unusual behavior of an operator can be flagged through the use of an exception. PSDL facilitates this through the use of the built-in abstract data type of Exception. Exceptions are identified by name. PSDL provides for the raising of an exception of a given name, detecting the presence of an exception with a given name, and determining if no exception was raised (i.e., Normal).

When an exception is raised by an operator, the exception is transmitted on all data streams of type exception, regardless of the exception stream label, leaving the operator, subject to the stream output guard. The exception is transmitted only over local exception streams. Thus the exception is not transmitted over exception streams which are outputs of another operator.
At least one exception output stream should be provided for each operator which is capable of generating an exception.

Operator Exceptions Popup Window

This window facilitates the entry of an operator's Exception constraint options through the use of a scrollable text widget. Any number of Exceptions can be specified for the selected operator.

The grammar for an Exception is as follows:

```
"exception" id ["if" expression] [reqmts_trace]
```

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information.

User options:

OK	-- Accept the text in the scrollable window and return to the Operator Property window.
Cancel	-- Abandon any changes made to the scrollable window and return to the Operator Property window.
Help	-- This display.

ID List Popup Window

This window facilitates the viewing, entry and modification of an identifier list. The window scrolls up and down for viewing a long lists.

Entry into the list is performed using the Add button. Move to the desired location and press the Add button. A prompt popup window will be provided for the entry of the new identifier. Select OK to accept the identifier and return to the identifier list.

An identifier is modified by selecting (highlighted) the desired identifier and pressing the Edit button. A prompt popup window will be provided similar to that used for adding an identifier. Select OK after correcting the identifier to accept the changes and return to the identifier list. If Edit is depressed without a selected identifier, a notice popup window will inform you that nothing was selected to edit.

An identifier is deleted by selecting (highlighted) the desired identifier and pressing the Delete button. The selected identifier will be removed from the list. If Delete is depressed without a selected identifier, a notice popup window will inform you that nothing was selected to delete.

Note that all modifications made to the identifier list are local to the scrollable window until the user accepts the list by depressing OK to return to the parent window. Cancel can be used to abandon all modifications made to the list.

Refer to the "PSDL Grammar" option under the "Help" menu bar for the syntax of an identifier.

User options:

OK	-- Accept the list of identifiers contained in the scrollable window and return to the parent window.
Cancel	-- Abandon any changes made to the list of identifiers and return to the parent window.
Add	-- Provide a prompt popup window for adding an identifier to the list.
Delete	-- Remove the identifier which the user has selected (highlighted).
Edit	-- Provide a prompt popup window for editing the identifier selected (highlighted) by the user.
Help	-- This display.

Prompt Popup User options:

OK	-- Accept the identifier and add it to the list of identifiers within the scrollable ID List window if Add was selected or replace the selected identifier within the scrollable ID List window if Edit was selected. Control is returned to the ID List popup window.
Cancel	-- Abandon the new identifier if Add was selected or any changes made to the identifier if Edit was selected. Control is returned to the ID List popup window.

Informal Design Description Popup Window

This window facilitates the entry of the graph's informal design description. The informal description provides for the free-format description of the graph. All printable characters are permitted except for '}'.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information.

User options:

OK	-- Accept the text in the scrollable window and return to the main window.
Cancel	-- Abandon any changes made to the scrollable window and return to the main window.
Help	-- This display.

Stream Initial Value Popup Window

This window facilitates the entry of a state stream's initial value expression through the use of a scrollable text widget. This option is only available for state streams.

Refer to the "PSDL Grammar" option under the "Help" menu bar for the syntax rules for an initial value.

User options:

OK	-- Accept the text in the scrollable window and return to the Stream Property window.
Cancel	-- Abandon any changes made to the scrollable window and return to the Stream Property window.
Help	-- This display.

Formal Design Description Popup Window

This window facilitates the entry of an operator's formal design description. The formal description provides for the free-format description of the operator. All printable characters are permitted except for '}'.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information.

User options:

OK	-- Accept the text in the scrollable window and return to the Operator Property window.
Cancel	-- Abandon any changes made to the scrollable window and return to the Operator Property window.
Help	-- This display.

Informal Design Description Popup Window

This window facilitates the entry of an operator's informal design description. The informal description provides for the free-format description of the operator. All printable characters are permitted except for '}'.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information.

User options:

OK	-- Accept the text in the scrollable window and return to the Operator Property window.
Cancel	-- Abandon any changes made to the scrollable window and return to the Operator Property window.
Help	-- This display.

Operator Property Popup Window

This window provides for the viewing and editing of a PSDL operator. All modifications to the operator are local to the Operator Property window until the operator is accepted by selecting OK. Changes can be abandoned by selecting Cancel.

Name: provides a text window from which the user can enter/modify the operator's name. The name must conform to the syntax of a PSDL identifier. The name must all be unique to the current data flow diagram.

Next to the operator's name is a pull down menu for selecting between an operator and a terminator. A terminator has a Maximum Execution Time (MET) of 0.

Implementation Language: provides a pull down menu from which the user may select the desired implementation language.

Trigger: provides for the specification of an operator's trigger control constraint. The syntax for a trigger is given by:

"triggered" [trigger] ["if" expression] [reqmts_trace]

where [trigger] is one of the following:

"by all" id_list
"by some" id_list

The first control provides for the specification of the [trigger], through the use of a pull down menu. By selecting "By Some" or "By All", a control widget is displayed for entering the Stream list.

Selection of the Stream List control will access an ID List popup window from which a list of stream identifiers can be specified.

A popup window can be accessed for specifying the trigger if condition expression by selecting the "If Condition" control.

Finally, the selection of the "Required By" control access an ID List popup window from which a list of requirement identifiers can be specified.

Timing: Access a pull down menu for selecting the timing characteristics of the operator. By default the operator is "Non-Time Critical". In this state, all additional timing parameters are unavailable. By selecting "Sporadic", the user is provided with the options of Maximum Execution Time (MET), Minimum Calling Period (MCP), and Maximum Response Time (MRT). By selecting "Periodic", the user is provided with the options of MET, Period, and Finish Within.

Each of timing options can be provided with a positive integer along with a time unit, which is available from a pull down menu. The "Required By" control access an ID List popup window from which a list of requirement identifiers can be specified.

The next three controls, "Output Guards", "Exceptions", and "Timer Ops" provide for the specification of operator control constraints.

There is a horizontal line separating the next three controls. These controls are used to enter data into the operator's specification.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information regarding the syntax of an operator.

User options:

OK	-- Accept the operator and return to the main window.
Cancel	-- Abandon any changes made to the operator and return to the main window.
Help	-- This display.

Operators:

A PSDL operator is a state machine. A state machine contains a finite number of inputs, outputs, and state variables; each of which are represented as a data stream.

When the state machine is executed, it reads one data object from each of the input streams. The output values of the state machine depend solely upon the current values of the input objects that were read and the current value of the state variables. At most, one data object is written to each output stream.

An operator that is implemented using a PSDL supported programming language is referred to as atomic. In this case, the operator's implementation specifies the programming language as well as the module name used to implement the operator. Operators that are not atomic are composite. A composite operator is itself decomposed into a network of operators; communicating through data streams. This establishes a parent-child relationship between operators. The composite operator being the parent and the operators contained in the decomposed network being the children.

Composite operators provide for a hierarchical decomposition of a prototype. At the top most level, the prototype consists of a single operator, referred to as the root operator. Children of the root operator can either be implemented as atomic operators or as composite operators. At the lowest level, all operators are implemented as atomic operators.

In support of real-time prototypes, PSDL provides both time critical and non-time critical operators. Execution of time critical operators can be triggered either periodically or sporadically (i.e., data driven). In order for the CAPS Execution Support subsystem to obtain a schedule which executes each operator consistently with the timing constraints of the augmented data flow diagram, a bound must be placed on the execution time of the operators. This bound is referred to as the Maximum Execution Time (MET). An operator is time critical if and only if it has been assigned an MET. Otherwise, the operator is non-time critical.

A time critical operator is triggered periodically if it contains a Period (P) timing constraint. Otherwise, the operator is triggered sporadically, based on the arrival of data (i.e., data driven), and must contain a data trigger control constraint.

Since a PSDL prototype represents a closed system, often it is necessary to include operators which are not considered to be part of the prototype to create the closed system. PSDL facilitates the inclusion of these external systems as terminators. Terminators are operators with an assigned MET of zero, and thus the time required to execute the terminator is not counted against the prototype execution time. The CAPS maintains a simulated real-time clock. During the execution of a terminator, the simulated real-time clock is turned off. Terminators are represented in the PSDL Editor by a rectangular bubble within the data flow diagram. Operators with a non-zero MET (including those operators with an undefined MET) are represented in the PSDL Editor by a circular bubble.

Operator Output Guard Popup Window

This window facilitates the entry of an operator's Output guard constraint options through the use of a scrollable text widget. Any number of Ouput guards can be specified for the selected operator.

The grammar for an Output guard is as follows:

```
"output" id_list "if" expression [reqmts_trace]
```

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information.

User options:

OK	-- Accept the text in the scrollable window and return to the Operator Property window.
Cancel	-- Abandon any changes made to the scrollable window and return to the Operator Property window.
Help	-- This display.

Printer Popup Window

This window facilitates the saving of the current data flow diagram as an output image. The image can be saved either to a printer to a file, by selecting either the Printer or the File radio button widgets.

If Printer is selected, the user can provide a printer name. If no printer name is supplied, the image is printed on the standard lpr printer. The printer must be capable of accepting PostScript input.

If File is selected, the data flow diagram is saved to the file specified by the user using the utility xwd. This command produces an X Window System Dump File formatted image. The file name must be provided.

User options:

OK	-- Save the image to either the printer or the user specified file.
Cancel	-- Return to the main window without saving the image.
Help	-- This display.

```
-- $Header: /work/berzins/TRANSLATOR/Documentation/RCS/psdl.grammar.current,
-- v 1.7 1996/08/03 18:15:56 berzins Exp berzins $
8/14/96 Valdis Berzins
Fix the definition of letter and alphanumeric lexical classes.

        Affected productions:
letter = "a .. z" | "A .. Z" | "_"
alpha_numberic = letter | digit

7/25/96 Valdis Berzins
Generalize implementation language id.

        Affected productions:
operator_impl = "implementation ada" op_name "end"

7/25/96 Valdis Berzins
        Fix unique-id suffix conventions for psdl identifiers.
Graph nodes have unique integer suffixes,
added by the psdl editor and invisible to the user.

        Affected productions:
op_name = ada_op_name "_" integer_literal
ada_op_name = id "_" integer_literal
op_id = [id "."] op_name ["(" [id_list] "|" [id_list] ")"]
operator_impl = "implementation ada" ada_op_name "end"

2/2/95 Valdis Berzins
       Correct typos: "initial_expression" --> "expression"
       in the descendents of "expression".

       Affected productions:
expression = initial_expression binary_op initial_expression
           | unary_op initial_expression

9/28/94: Valdis Berzins
        Augment vertex and edge decelarations to include
        symbolic property lists, to support a stable interface
        between the GE and the rest of CAPS.

        Affected productions:
vertex = "vertex" op_id [":" time]
edge = "edge" id [":" time] op_id "->" op_id

        New productions:
property = "property" id "=" expression

7/28/94: Valdis Berzins
        Change op_id to allow adt operations to appear in the graph.

        Affected productions:
op_id = id ["(" [id_list] "|" [id_list] ")"]

        Restrict id's representing ada operator names to include an
        integer suffix representing a unique id. Restrict psdl operator
        names to include two integer suffixes, the first representing
        a unique id and the second representing an instance number.
        The instance number is zero for all operator declarations,
        and is nonzero for graph vertices, to represent operations
        of a psdl type that appear more than once in the expanded graph.

        New productions:
op_name = ada_op_name "_" integer_literal
ada_op_name = id "_" integer_literal

        Affected productions:
type_spec = "specification" ["generic" type_decl] [type_decl]
            {"operator" id operator_spec} [functionality] "end"
operator = "operator" id operator_spec operator_impl
type_impl = "implementation" type_name {"operator" id operator_impl} "end"
operator_impl = "implementation ada" id "end"
initial_expression = type_name "." id ["(" initial_expression_list ")"]
expression = type_name "." id ["(" expression_list ")"]
op_id = id ["(" [id_list] "|" [id_list] ")"]

11/14/91: Change entered by Charlie Altizer.
        The phrase "BY REQUIREMENTS" was changed to "REQUIRED BY."

        Affected productions:
                reqmts_trace = "by requirements" id_list

psdl grammar 12/1/90
end Header

Optional items are enclosed in [ square brackets ].  Items which may
appear zero or more times appear in { braces }.  Terminal symbols appear
in " double quotes ".  Groupings appear in ( parentheses ).

*************************************************************************

psdl
        = {component}

component
        = data_type
        | operator

data_type
        = "type" id type_spec type_impl

type_spec
        = "specification" ["generic" type_decl] [type_decl]
          {"operator" op_name operator_spec}
          [functionality] "end"

operator
        = "operator" op_name operator_spec operator_impl
```

```
operator_spec
        = "specification" {interface} [functionality] "end"

interface
        = attribute [reqmts_trace]

attribute
        = "generic" type_decl
        | "input" type_decl
        | "output" type_decl
        | "states" type_decl "initially" initial_expression_list
        | "exceptions" id_list
        | "maximum execution time" time

type_decl
        = id_list ":" type_name {"," id_list ":" type_name}

type_name
        = id
        | id "[" type_decl "]"

id_list
        = id {"," id}

reqmts_trace
        = "required by" id_list

functionality
        = [keywords] [informal_desc] [formal_desc]

keywords
        = "keywords" id_list

informal_desc
        = "description" "{" text "}"

formal_desc
        = "axioms" "{" text "}"

type_impl
        = "implementation" id id "end"
        | "implementation" type_name {"operator" op_name operator_impl} "end"

operator_impl
        = "implementation" id id "end"
        | "implementation" psdl_impl "end"

psdl_impl
        = data_flow_diagram [streams] [timers] [control_constraints]
          [informal_desc]

data_flow_diagram
        = "graph" {vertex} {edge}

vertex
        = "vertex" op_id [":" time] {property}
          -- time is the maximum execution time

edge
        = "edge" id [":" time] op_id "->" op_id {property}
          -- time is the latency

property
        = "property" id "=" expression

op_id
        = [id "."] op_name ["(" [id_list] "|" [id_list] ")"]

streams
        = "data stream" type_decl

timers
        = "timer" id_list

control_constraints
        = "control constraints" constraint {constraint}

constraint
        = "operator" op_id
          ["triggered" [trigger] ["if" expression] [reqmts_trace]]
          ["period" time [reqmts_trace]]
          ["finish within" time [reqmts_trace]]
          ["minimum calling period" time [reqmts_trace]]
          ["maximum response time" time [reqmts_trace]]
          {constraint_options}

constraint_options
        = "output" id_list "if" expression [reqmts_trace]
        | "exception" id ["if" expression] [reqmts_trace]
        | timer_op id ["if" expression] [reqmts_trace]

trigger
        = "by all" id_list
        | "by some" id_list

timer_op
        = "reset timer"
        | "start timer"
        | "stop timer"

initial_expression_list
        = initial_expression {"," initial_expression}

initial_expression
        = "true"
```

```
        | "false"
        | integer_literal
        | real_literal
        | string_literal
        | id
        | type_name "." op_name ["(" initial_expression_list ")"]
        | "(" initial_expression ")"
        | initial_expression binary_op initial_expression
        | unary_op initial_expression

binary_op
        = "and" | "or" | "xor"
        | "<" | ">" | "=" | ">=" | "<=" | "/="
        | "+" | "-" | "&" | "*" | "/" | "mod" | "rem" | "**"

unary_op
        = "not" | "abs" | "-" | "+"

time
        = integer_literal unit

unit
        = "microsec"
        | "ms"
        | "sec"
        | "min"
        | "hours"

expression_list
        = expression {"," expression}

expression
        = "true"
        | "false"
        | integer_literal
        | time
        | real_literal
        | string_literal
        | id
        | type_name "." op_name ["(" expression_list ")"]
        | "(" expression ")"
        | expression binary_op expression
        | unary_op expression

op_name
        = id

id
        = letter {alpha_numeric}

real_literal
        = integer_literal "." integer_literal

integer_literal
        = digit {digit}

string_literal
        = """ {char} """

char
        = any printable character except "}"

digit
        = "0 .. 9"

letter
        = "a .. z"
        | "A .. Z"

alpha_numberic
        = letter
        | digit
        | "_"

text
        = {char}
```

Prototype Specification Popup Window

This window is primarily used to view the specification of a PSDL operator. In addition, there are a few constructs of the specification which can only be specified through this window. The remainder of the operator's specification is automatically generated by the background checker.

The input, output, and state list of the operator specification are provided by the background checker. The user is allowed to specify the generic and exception list as well as the maximum execution time.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information regarding the syntax of an operator's specification.

User options:

OK	-- Accept the text in the scrollable window and return to the main window.
Cancel	-- Abandon any changes made to the scrollable window and return to the main window.
Help	-- This display.

Stream Property Popup Window

This window provides for the viewing and editing of a data flow diagram stream. Both streams and state streams are supported by this popup window. All modifications to the stream are local to the Stream Property window until the stream is accepted by selecting OK. Changes can be abandoned by selecting Cancel.

Stream Name: provides a text window from which the user can enter/modify the stream's name. The name must conform to the syntax of a PSDL identifier.

Stream Type: provides a text window from which the user can enter/modify the stream's type. The name must conform to the syntax of a PSDL identifier.

State Stream: Yes specifies that the stream is a state stream. No specifies a normal stream.

Selection of state stream access the state stream initial value. Selection of the "State Initial Value" control access a scrollable text window for the specification of the initial value expression.

Latency: provides for the entry of a positive integer and a timing unit, through the use of a pull down menu, for specifying the latency in the stream.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information regarding the syntax of a stream.

User options:

OK	-- Accept the stream and return to the main window.
Cancel	-- Abandon any changes made to the stream and return to the main window.
Help	-- This display.

Streams:

Streams are used to communicate data objects of a fixed data type from a set of one or more producer operators to a set of one or more consumer operators. While the PSDL syntax and the PSDL Editor represent a stream as a link from one producer to one consumer, multiple producers and multiple consumers are supported by matching stream names (i.e., identifiers) within the stream scope.

PSDL provides two types of data streams: data flow streams and sampled streams. A data flow stream guarantees that no data object is lost or replicated. The data flow stream behaves like a first-in-first-out (FIFO) queue with a length of one for each consumer. A sampled stream behaves like a single memory cell which contains one data object for each consumer. The most recent data value is obtained each time the stream is read. Thus, data objects may be lost if associated with a fast producer or replicated if associated with a slow producer. The type of data stream used is determined by the consuming operator's trigger control constraint. If the consuming operator contains a "triggered by all" control constraint for a set of streams, those streams are data flow streams. Otherwise, the stream is a sampled stream.

The FIFO queue length of one for a data flow stream restricts the relative execution rates between the producer and consumer operators. In order to guarantee that no data object is lost or replicated, the output rate of the producer must not exceed the execution rate of the consumer for all data flow streams. No such restriction is placed on a sampled stream.

State Streams:

State streams provide a state machine with memory. State streams are also used to schedule data flow diagrams that would otherwise be impossible to schedule due to circular precedence constraints. A state stream is declared in the specification section of the component in which the state stream first appears in the data flow diagram.

A state stream differs from a data stream in that a state stream must include an initial value. It is the initial value of a state stream which makes it possible for a state stream to break a circular precedence constraint. A state stream is also required when connecting time critical and non-time critical operators.

Stream Consistency:

Within the hierarchical structure of PSDL, a composite operator is implemented as a data flow diagram of a PSDL component at a lower level. All inputs to the composite operator must be utilized as an external input to at least one of the operators in the decomposed data flow diagram. Likewise, all outputs of the composite operator must be utilized as an external output from at least one of the operators in the decomposed data flow diagram. A similar set of rules require that all external inputs to a decomposed data flow diagram must be inputs to the composite operator and all external outputs to a decomposed data flow diagram must be outputs from the composite operator.

External streams in a decomposed data flow diagram inherit the stream's data object type of the the composite operator. In addition, which type of stream (i.e., data flow stream or sample stream) is derived from the trigger constraint of the consuming operator. For a composite operator, as explained above, an input to a composite operator must also be an input to at least one operator in the decomposed data flow diagram. The trigger constraints of both the composite operator and those of the applicable decomposed operators are used in the derivation of the type.

Stream Data Flow Diagram Representation:

Streams are represented as directed lines within the data flow diagram, with the arrow pointing to the consumer operator. State streams are similarly represented. However, the state stream uses a bold directed line. External streams are missing either a producer or consumer operator. The missing operator is replaced with "External".

Operator Timers Popup Window

This window facilitates the entry of an operator's Timer constraint options through the use of a scrollable text widget. Any number of Timer constraints can be specified for the selected operator.

The grammar for a Timer constraint is as follows:

```
timer_op id ["if" expression] [reqmts_trace]
```

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information.

User options:

OK	-- Accept the text in the scrollable window and return to the Operator Property window.
Cancel	-- Abandon any changes made to the scrollable window and return to the Operator Property window.
Help	-- This display.

Timers:

PSDL provides timers as predefined abstract state machines. A timer behaves similar to a stopwatch. A timer is modeled as an elapsed time value and a run switch. As long as the run switch is on, the elapsed time value is incremented. Timers have four operations: start, stop, reset, and read. Start turns on the run switch. Stop turns off the run switch. Reset turns off the run switch and sets the elapsed time value to zero. Read returns the current value of the elapsed time.

Timers are declared in the implementation section of a composite operator. Timers differ from operators in that they do not appear in the data flow diagram. Timer values (i.e, the result of a read operator) are accessed by referring to the timer by name within the control constraints of an operator.

A timer is visible within the component in which it is declared. If the component's data flow diagram contains a composite operator, then the timer is visible within the decomposed components.

Timers Tool Popup Window

This window facilitates the viewing, entry and modification of a list of timers. The window scrolls up and down for viewing a long list of timers. Timers must conform to the syntax of a PSDL identifier.

Entry into the list is performed using the Add button. Move to the desired location and press the Add button. A prompt popup window will be provided for the entry of the new identifier. Select OK to accept the identifier and return to the identifier list.

An identifier is modified by selecting (highlighted) the desired identifier and pressing the Edit button. A prompt popup window will be provided similar to that used for adding an identifier. Select OK after correcting the identifier to accept the changes and return to the identifier list. If Edit is depressed without a selected identifier, a notice popup window will inform you that nothing was selected to edit.

An identifier is deleted by selecting (highlighted) the desired identifier and pressing the Delete button. The selected identifier will be removed from the list. If Delete is depressed without a selected identifier, a notice popup window will inform you that nothing was selected to delete.

Note that all modifications made to the identifier list are local to the scrollable window until the user accepts the list by depressing OK to return to the parent window. Cancel can be used to abandon all modifications made to the list.

Refer to the "PSDL Grammar" option under the "Help" menu bar for the syntax of an identifier.

User options:

OK	-- Accept the list of identifiers contained in the scrollable window and return to the main window.
Cancel	-- Abandon any changes made to the list of identifiers and return to the main window.
Add	-- Provide a prompt popup window for adding an identifier to the list.
Delete	-- Remove the identifier which the user has selected (highlighted).
Edit	-- Provide a prompt popup window for editing the identifier selected (highlighted) by the user.
Help	-- This display.

Prompt Popup User options:

OK	-- Accept the identifier and add it to the list of identifiers within the scrollable ID List window if Add was selected or replace the selected identifier within the scrollable ID List window if Edit was selected. Control is returned to the ID List popup window.
Cancel	-- Abandon the new identifier if Add was selected or any changes made to the identifier if Edit was selected. Control is returned to the ID List popup window.

Operator Trigger If Condition Popup Window

This window facilitates the entry of an expression for the operator's Trigger control constraint. The syntax for the Trigger control constraint is:

"triggered" [trigger] ["if" expression] [reqmts_trace]

where trigger is defined by one of following:

"by all" id_list
"by some" id_list

The "If Condition" button is used to provide the "expression" contained within the "If" term. The id_list of the trigger is specified by first selecting "by all" or "by some" in the Trigger pulldown menu. The id_list is not visible when the Trigger is "unprotected".

Use the scrollable window to specify the expression. Accept the expression by depressing the OK button and return to the Operator Property window.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information on the syntax of an "expression".

User options:

OK	-- Accept the text in the scrollable window and return to the Operator Property window.
Cancel	-- Abandon any changes made to the scrollable window and return to the Operator Property window.
Help	-- This display.

Prototype Types Specification Popup Window

This window facilitates the entry of the prototype's Types.

Refer to the "PSDL Grammar" option under the "Help" menu bar for additional information regarding the syntax of a type.

User options:

OK	-- Accept the text in the scrollable window and return to the main window.
Cancel	-- Abandon any changes made to the scrollable window and return to the main window.
Help	-- This display.

APPENDIX E. INSTALLATION

The PSDL Editor consists of the background checker and the graph editor. This appendix describes the process required to compile the graph editor as well as a PSDL Editor driver, used for debugging the graph editor.

1. SOFTWARE REQUIREMENTS

Table XIX contains the software and version numbers used to generate the graph editor. The graph editor was developed on a Sun workstation.

2. COMPILING THE GRAPH EDITOR

Contained within the graph editor source code is the file `Makefile`, used to build the graph editor. This file is configured to be executed from the graph editor source code directory, which is a subdirectory of a PSDL Editor directory. Prior to generating the graph editor, the user should set the default directory to that which contains the graph editor source code. The graph editor can be generated by simply typing "`make`" at the Unix prompt. If the graph editor compiles successfully, the image `edit_graph` will be generated. This file will automatically be located in the parent directory. In addition, the image `sde` will also be generated, in the parent directory. This is the PSDL Editor driver program, used to test the graph editor in

Table XIX. Support Software

Software	Version
Operating System	SunOS Release 4.1.3
C Compiler	`gcc` Version 2.6.3
C++ Compiler	`g++` Version 2.6.3
Windows	`X11` Version 5
	`Motif` Version 1.1
Make	Sun Version 4.1

Table XX. Graph Editor Required Files

edit_graph	output_guard.hlp
error.hlp	print.hlp
exceptions.hlp	psdl_grammar.hlp
exceptions_list.hlp	spec_tool.hlp
id_list.hlp	stream_property.hlp
inform_tool.hlp	streams.hlp
initial_state.hlp	timer_list.hlp
op_prop_formal_desc.hlp	timers.hlp
op_prop_informal_desc.hlp	timers_tool.hlp
operator_property.hlp	trigger_if_cond.hlp
operators.hlp	types_tool.hlp

a standalone fashion.

Table XX provides a list of the graph editor files required to support the PSDL Editor. The file `edit_graph` is the image which executes the graph editor. The remainder of the files are help files, which are expected to be located in the directory from which the graph editor is executed.

3. X WINDOW SYSTEM CUSOMIZATION

The PSDL Editor uses Motif to build the graph editor. Consequentially, the X Window System initialization protocol used to define window parameters can be used with the PSDL Editor [Ref. 15]. These parameters can be defined in the file `.Xresources-color`[29], located in your home directory[30] Table XXI provides sample declarations to set the default window size for the graph editor as well as for defining the delete key to work like the backspace key.

[29]Or in the file `.Xresources-mono` for non-color systems.

[30]Since this is a "dot" file, it will not appear under the Unix "`ls`" command. "Dot" files can be seen by using the `-a` flag (e.g., "`ls -a`").

Table XXI. X Window System Initialization

```
edit_graph*geometry: =900x700
edit_graph*XmText.translations: #override\n\
        <Key>osfDelete: delete-previous-character()
edit_graph*XmTextField.translations: #override\n\
        <Key>osfDelete: delete-previous-character()
```

LIST OF REFERENCES

[1]Luqi and Mohammad Ketabchi. A computer-aided prototyping system. *IEEE Software*, pages 66–72, March 1988.

[2]Luqi and Man-Tak Shing. Teaching hard real-time software development via prototyping. In *Proceedings of the ACM/IEEE International Workshop on Software Engineering Education*, pages 199–211, Sorrento, Italy, May 1994. ACM/IEEE.

[3]Luqi, Valdis Berzins, and Raymond T. Yeh. A prototyping language for real-time software. *IEEE Transactions on Software Engineering*, 14(10):1409–1423, October 1988.

[4]Christopher S. Eagle. Tools for storage and retrieval of ada software components in a software base. Master's thesis, Naval Postgraduate School, Monterey, CA, March 1995.

[5]Valdis Berzins and David Dampier. Software merge: Combining changes to decompositions. *Journal of Systems Integration*, 6:135–150, 1996.

[6]Kyongsuk Pace. CAPS 93 tutorial. NPS CAPS Project, May 1996.

[7]Luqi. Guest editor's introduction. *Journal of Systems Integration*, 6:15–17, 1996.

[8]Thomas W. Reps and Tim Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors*. Springer-Verlag, 1988.

[9]Luqi. Cs4920: Computer aided prototyping systems. Course notes from NPS CS4920 AY96Q3, 1996.

[10]Bernd Krämer, Luqi, and Valdis Berzins. Compositional semantics of a real-time prototyping language. *IEEE Transactions on Software Engineering*, 19(5):453–477, May 1993.

[11]Valdis Berzins and Luqi. *Software Engineering with Abstractions*. Addison-Wesley Publishing Company, 1991.

[12]Inernational Organization for Standardization. *Ada Reference Manual*, November 1994.

[13]Robert Mobley Dixon. The design and implementation of a user interface for the computer-aided prototyping system. Master's thesis, Naval Postgraduate School, Monterey, CA, September 1992.

[14]Ravi Sethi. *Programming Languages: Concepts and Constructs*. Addison-Wesley Publishing Company, second edition, 1996.

[15]Valerie Quercia and Tim O'Reilly. *X Window System User's Guide: OSF/Motif 1.2 Edition*, volume 3. O'Reilly and Associates, Inc., 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Dr. Ted Lewis, Chairman, Code CS/Lt 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5100

4. Dr. Valdis Berzins, Code CS/Be 3
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5100

5. Dr. Luqi, Code CS/Lq 4
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5100

6. Dr. Man-Tak Shing, Code CS/Sh 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5100

7. Peter K. Burke 1
   412 Test Wing/TSVE
   195 E. Popson Ave.
   Edwards AFB, CA 93523

8. Kenneth Moeller 1
   1033 East Glenn Ct
   Lancaster, CA 93535